



*Instruction Sets Guide*

*Rev. 21.0*

*DOC9474-2LA*

# Instruction Sets Guide

Second Edition

by  
Marilyn Hammond

Prime Computer, Inc.  
Prime Park  
Natick, Massachusetts 01760

The information in this document is subject to change without notice and should not be construed as a commitment by Prime Computer, Inc. Prime Computer, Inc., assumes no responsibility for any errors that may appear in this document.

The software described in this document is furnished under a license and may be used or copied only in accordance with the terms of such license.

Copyright © 1987 by Prime Computer, Inc. All rights reserved.

PRIME and PRIMOS are registered trademarks of Prime Computer, Inc. DISCOVER, INFO/BASIC, INFORM, MIDAS, MIDASPLUS, PERFORM, Prime INFORMATION, PRIME/SNA, PRIMELINK, PRIMENET, PRIMEWAY, PRIMIX, PRISAM, PST 100, PT25, PT45, PT65, PT200, PW153, PW200, PW250, RINGNET, SIMPLE, 50 Series, 400, 750, 850, 2250, 2350, 2450, 2550, 2650, 2655, 2755, 6350, 9650, 9655, 9750, 9755, 9950, 9955, and 9955II are trademarks of Prime Computer, Inc.

#### PRINTING HISTORY

First Edition (DOC9474-11A) January 1985  
Update 1 (UPD9474-11A) October 1985  
Update 2 (UPD9474-12A) February 1986  
Update 3 (UPD9474-13A) April 1986  
Second Edition (DOC9474-21A) August 1987

#### CREDITS

Editorial: Thelma Henner  
Project Support: The CPU Group  
Illustration: Mingling Chang  
Document Preparation: Kathy Normington  
Production: Judy Gordon

## HOW TO ORDER TECHNICAL DOCUMENTS

To order copies of documents, or to obtain a catalog and price list:

### United States Customers

Call Prime Telemarketing,  
toll free, at 1-800-343-2533,  
Monday through Friday,  
8:30 a.m. to 5:00 p.m. (EST).

### International

Contact your local Prime  
subsidiary or distributor.

## CUSTOMER SUPPORT

Prime provides the following toll-free numbers for customers in the United States needing service:

1-800-322-2838 (within Massachusetts)	1-800-541-8888 (within Alaska)
1-800-343-2320 (within other states)	1-800-651-1313 (within Hawaii)

For other locations, contact your Prime representative.

## SURVEYS AND CORRESPONDENCE

Please comment on this manual using the Reader Response Form provided in the back of this book. Address any additional comments on this or other Prime documents to:

Technical Publications Department  
Prime Computer, Inc.  
500 Old Connecticut Path  
Framingham, MA 01701



# Contents

ABOUT THIS BOOK	vii
1 INTRODUCTION	
Addressing Modes	1-1
Summary of Datatypes and Applicable Instructions	1-5
2 S, R, AND V MODE	
Introduction	2-1
Instructions	2-7
3 I MODE	
Introduction	3-1
Instructions	3-7
APPENDICES	
A Condition Code Information	A-1
B Addressing Information	B-1
Addressing Modes and Formats	B-1
Address Traps	B-18
Summary	B-22
C Instruction Summary Charts	C-1
D Hardware Considerations in Performance	D-1
Instruction Weights	D-2
Extensions to Instruction Weights	D-7
E Archived Instructions	E-1
F 2455 Instruction Sets	F-1



# About This Book

Prime's 50 Series™ family is a sophisticated group of totally compatible supermini computers. Its members are the Prime:

6350™	9955 II™	9955™	9950™
9755™	9750™	9655™	9650™
2755™	2655™	2550™	2450™
2350™	2250™	850™	750™
650™	550-II™	550™	500™
450™	I450™	400™	350™
250-II™	250™	150™	

The earlier processors are the 2250, 850, 750, 650, 550-II, 550, 500, 450/, I450, 400, 350, 250-II, 250, and 150.

The 50 Series systems embody an advanced 32-bit architecture that grants the user the ability to perform complex tasks efficiently and quickly. This document describes the 50 Series addressing modes and their instructions from a functional point of view.

## NOTES TO THE READER

Several groups of people will find this document useful: engineers, programmers, designers, and technicians. To read this book, you should have a basic understanding of computers, but not necessarily of Prime computers. Prime stresses a high degree of compatibility across its product line; therefore, you can apply much of the information contained in this book to other Prime machines, as well as to the 50 Series machines.

## ORGANIZATION OF THIS GUIDE

This guide describes the instructions for S, R, V, and I addressing modes. Each of these modes is introduced in Chapter 1. This chapter also presents the 50 Series datatypes and their applicable instructions. Chapters 2 and 3 contain detailed information about each instruction — name, format, mnemonic, and required operands -- and a complete description of each of the instruction's actions.

Chapters 1 through 3 may be summarized as follows:

- Chapter 1 contains brief descriptions of S, R, V, and I addressing modes as well as a summary of datatypes with applicable instructions.
- Chapter 2 is a dictionary of instructions executable in S, R, and V modes.
- Chapter 3 is a dictionary of instructions executable in I mode.

Appendix A discusses the condition codes and their interpretation.

Appendix B presents tables of addressing information.

Appendix C contains summary charts of the instructions.

Appendix D discusses hardware considerations in performance and provides tables of relative instruction weights.

Appendix E has those instructions that have been archived.

Appendix F discusses the instructions sets in relation to the 2455.

## PRIME DOCUMENTATION CONVENTIONS

The following conventions are used in command formats, statement formats, and in examples throughout this document. Examples illustrate the uses of these commands and statements in typical applications.

<u>Convention</u>	<u>Explanation</u>	<u>Example</u>
UPPERCASE	In command formats, words in uppercase indicate the names of commands, options, statements, and keywords. Enter them in uppercase.	CRL
lowercase	In command formats, words in lowercase indicate variables for which you must substitute a suitable value.	LDA address
Brackets [ ]	Brackets enclose an optional item.	[ DISPLACEMENT\16 ]
Apostrophe '	An apostrophe preceding a number indicates that the number is in octal.	'200

# 1

## Introduction

This chapter briefly describes the S, R, V, and I addressing modes as well as introducing their data representations. Each datatype operation is listed with its S, R, V, and I mode instructions.

### ADDRESSING MODES

The 50 Series processors support four addressing modes, each of which forms addresses differently. Depending on the program and personal preference, one or two of these modes may be more useful than another. The three most important modes are:

- V, or virtual
- I, or general register
- R, or relative

The fourth mode -- S, or sector, mode -- is supported for historical reasons.

V Mode

V mode performs short and long operations and has a wide variety of registers to use. A short (16-bit) instruction in this mode can reference the first 256 locations of both the stack and link, as well as the 224 locations on either side of the current location in the procedure segment. A long (32-bit) V mode instruction can directly reference all locations in four segments. Indirect addressing can reference all locations in up to 4096 128-Kbyte segments.

I Mode

When referencing memory, I mode is similar to 32-bit V mode. The difference is that I mode short operations reference 8 32-bit general purpose registers for use as index registers, accumulators, counters, or the like. I mode long operations have the same referencing power as V mode long operations. They can also use immediate forms and five additional index registers. (This makes a total of 7 index registers that I mode long operations can use.) The index registers are specified by the source register field. General register 0, however, cannot be used for indexing.

General register relative (GRR) is an addressing capability added to 32I mode that speeds up big array accesses and often gives the effect of using general registers as base registers. (This is sometimes called IX mode.) The offset is formed in GRR by adding the displacement to bits 17 to 32 of the source register field. GRR is used by the I mode instructions AIP and LIP. (GRR is not available for the earlier processors listed in "About This Book".)

The C language pointer is used by the I mode instructions ACP, CCP, DCP, ICP, LOC, SOC, and TCNP. The format of this pointer is the same as the indirect pointer, except that bit 4 is redefined as the B (byte) bit. When this bit contains 0, it indicates that bits 1 to 8 (the left byte) of an address contain the character to be used; when this bit contains 1, bits 9 to 16 (the right byte) of an address contain the character. A null pointer is represented by a 0 in bits 4 through 32. (The C language pointer and its instructions are not available for the earlier processors listed in "About This Book".)

Normal effective address formation uses either a base register, indirect pointer (IP) or a general register (for GRR addressing) as the source of the ring field, B bit, and segment number. The C language pointer is well defined for the IP and GRR form. When the base register is the source of the B bit, software depends on finding it reset to zero, pointing to the leftmost byte. While it is possible to set the E bit in a base register using 48-bit IPs to specify 32-bit addresses, this practice is not now done. Future implementations of V and I modes will force bit 4 to zero during effective address formation when the source of the segment is a base register; otherwise it will copy bit 4.

R Mode

A sector is a block of 512 (1000 octal) contiguous memory locations. Sector 0 starts on location 0 and ends on location '777; Sector 1 begins on location '1000 and ends on location '1777; and so on.

An R mode instruction can reference any location in Sector 0, as well as a group of locations relative to the current value of the program counter. When the sector bit (S) in an R mode instruction is 0, the instruction can only reference locations in Sector 0. When S is 1, the instruction references locations relative to the current value of the program counter. The range of these relative locations is PC - '360 to PC + '377, inclusive.

Note that an R mode instruction that specifies a location in the range PC - '361 to PC - '400, inclusive, selects a special addressing code, such as stack register.

S Mode

Like R mode instructions, S mode instructions contain a sector bit. When S is 0, references are to Sector 0. When S is 1, however, references are only to those locations within the sector containing the instruction.

S mode is a holdover from early Prime machines that were based on the Honeywell 316 and 516 minicomputers. When operating in S mode, the 50 Series processors act exactly as these early machines do.

Summary of Addressing Modes

Table 1-1 summarizes addressing information about S, R, V, and I modes. For further information, see Chapter 3 of the System Architecture Reference Guide.

Table 1-1  
Summary of Addressing Modes

Mode	Address Length	Addressing Range	# Index Regs	Indirection Levels
16S direct	14 bits	1024 halfwords	One	
16S indirect	14 bits	16K halfwords	One	Multiple
32S direct	15 bits	1024 halfwords	One	
32S indirect	15 bits	32K halfwords	One	Multiple
32R direct	15 bits	1008 halfwords	One	
32R indirect	15 bits	32K halfwords	One	Multiple
64R direct	16 bits	1008 halfwords	One	
64R indirect	16 bits	64K halfwords	One	One
64V 16-bit instructions	16 bits	64K halfwords: +256 SB relative +256 LB relative +/-256 PC relative +512 PB absolute	One	One
64V 32-bit instructions	28 bits	4 segments*	Two	One
64V indirect	28 bits	4096 segments*	Two	One
32I all	28 bits	12 segments* with GRR**	Seven	One
32I indirect	28 bits	4096 segments*	Seven	One

\* All segments contain 128 Kbytes.

\*\* Four segments for the 2250 and earlier processors because they have no GRR capability.

SUMMARY OF DATATYPES AND APPLICABLE INSTRUCTIONS

The 50 Series systems support several data representations. These representations fall into the major groups:

- Fixed-point data
- Floating-point numbers
- Decimal integers
- Character strings
- Queues

Tables 1-2 and 1-3 list the instructions applicable to the datatype operations (other than queues) available in S, R, V, and I modes. The body of each table shows which instructions perform a specific operation on a specific datatype. For detailed information about each instruction, refer to the instruction dictionaries in Chapters 2 and 3 of this manual. For further information about datatypes, see Chapter 6 of the System Architecture Reference Guide.

When using Tables 1-2 and 1-3, aa represents the set of arithmetic conditions [ EQ, GE, GT, LE, LT, NE ]. Also, these tables do not include instructions that operate on CBIT, LINK, the condition codes, or queues.

Throughout the rest of this book, R is used to indicate a 32-bit I mode general register, while r indicates bits 1-16 of a 32-bit I mode general register. In addition, A and B represent the S and R mode 16-bit registers; L and E represent the V mode 32-bit registers.

Table 1-2  
Summary of Datatypes and Applicable S, R, V Mode Instructions

Operation	Size of Datatype (in Bits of Register)							
	16 (A)	31 (A/B)	32 (L)	64 (L/E)	32FP (FAC)	64FP (DAC)	128FP (QAC)	Dec (-)
Load from memory	LDA	DLD	LDL		FLD	DFLD	QFLD	XMV
Store to memory	STA	DST	STL		FST	DFST	QFST	
Add	ADD	DAD	ADL		FAD	DFAD	QFAD	XAD
Subtract	SUB	DSB	SBL		FSB	DFSB	QFSB	XAD
Multiply	MPY		MPL		FMP	DFMP	QFMP	XMP
Divide	DIV		DVL		FDV	DFDV	QFDV	XDV
Increment	IRS, A1A, A2A							
Decrement	S1A, S2A							
AND	ANA		ANL					
OR	ORA							
XOR	ERA		ERL					
Complement	CMA							
Compare	CAS, CAZ		CLS		FCS	DFCS	QFC, QFCS	XCM
Logical test	Laa		LLaa		LFaa	LFaa		
Branch	Baa		BLaa		BFaa	BFaa		
Logical left shift	ALL		LLL					
Logical right shift	ARL		IRL					
Arithmetic left shift	ALS	LLS	LLS					
Arithmetic right shift	ARS	LRS	LRS					
Rotate left shift	ALR		LIR					

Table 1-2 (continued)  
Summary of Datatypes and Applicable S, R, V Mode Instructions

Operation	Size of Datatype (in Bits of Register)							
	16 (A)	31 (A/B)	32 (L)	64 (L/E)	32FP (FAC)	64FP (DAC)	128FP (QAC)	Dec (-)
Rotate right shift	ARR		LRR					
Clear	CRA	CRL	CRL	CRLE				
Clear left	CAL	CRA	CRA	CRL				
Clear right	CAR	CRB	CRB	CRE				
Interchange halves	ICA	IAB	IAB	ILE				
Interchange and clear left	ICL	XCA	XCA					
Interchange and clear right	ICR	XCB	XCB					
Two's complement	TCA		TCL		FCM	DFCM	QFCM	
Set sign	SSM	SSM	SSM					
Clear sign	SSP	SSP	SSP					
Change sign	CHS		CHS					
Convert datatypes:								
Integer to floating point	FLTA	FLOT	FLTL					
Floating point to integer	INTA	INT	INTL				QINQ QIQR	
Binary to decimal	XBTD		XBTD	XBTD				
Decimal to binary	XDTB		XDTB	XDTB				
Position for integer divide	PIDA	PID	PIDL	PIDL				
Position after multiply	PIMA	PIM	PIML	PIML				
Skips	Saa				FSaa	FSaa		

Table 1-3  
Summary of Datatypes and Applicable I Mode Instructions

Operation	Size of Datatype (in Bits of Register)						
	16 (r)	32 (R)	64 (R/R+1)	32FP (FAC)	64FP (DAC)	128FP (QAC)	Dec (-)
Load from memory	LH	L		FL	DFL	QFLD	XMV
Store to memory	STH	ST		FST	DFST	QFST	
Add	AH	A		FA	DFA	QFAD	XAD
Subtract	SH	S		FS	DFS	QFSB	XAD
Multiply	MH	M		FM	DFM	QFMP	XMP
Divide	DH	D		FDV	DFDV	QFDV	XDV
Increment	IMH, IH1, IH2	IM, IR1, IR2					
Decrement	DMH, DH1, DH2	DM, DR1, DR2					
AND	NH	N					
OR	OH	O					
XOR	XH	X					
Complement	CMH	CMR					
Compare	CH	C		FC	DFC	QFC,	XCM
Logical test	LHaa	Laa		LFaa	LFaa		
Branch	BHaa	BRaa		BFaa	BFaa		
Logical shift		SHL					
Arithmetic shift		SHA					
Shift right 1	SHR1	SR1					
Shift right 2	SHR2	SR2					
Shift left 1	SHL1 LHL1	SL1					

Table 1-3  
Summary of Datatypes and Applicable I Mode Instructions

Operation	Size of Datatype (in Bits of Register)						
	16 (r)	32 (R)	64 (R/R+1)	32FP (FAC)	64FP (DAC)	128FP (QAC)	Dec (-)
Shift left 2	SHL2 LHL2	SL2					
Shift left 3	LHL3						
Rotate		ROT					
Clear		CR					
Clear left	CRBL	CRHL					
Clear right	CRBR	CRHR					
Interchange halves	IRB	IRH	I				
Interchange and clear left	ICBL	ICHL					
Interchange and clear right	ICBR	ICHR					
Two's complement	TCH	TC		FCM	DFCM	QFCM	
Set sign	SSM	SSM					
Clear sign	SSP	SSP					
Change sign	CHS	CHS					
Convert datatypes: Integer to floating point	FLTH	FLT					
Floating point to integer	INTH	INT				QINQ QIQR	
Binary to decimal	XBTD	XBTD	XBTD (DAC)				
Decimal to binary	XDTB	XDTB	XDTB (DAC)				
Position for integer divide	PIDH	PID	PID				
Position after multiply	PIMH	PIM	PIM				

# 2

## S, R, and V Mode

### INTRODUCTION

This chapter contains descriptions for all 50 Series instructions used in S, R, and V modes. In the description of each instruction, you will find:

- The instruction mnemonic followed by any arguments.
- The name of the instruction.
- The bit format of the instruction.
- The modes for which the instruction is valid.
- Detailed information describing the instruction's action.
- Information about the how the instruction affects LINK, CBIT, and the condition codes.

### Notation Conventions

Several abbreviations and symbols are used throughout this dictionary. Table 2-1 defines the dictionary notation.

Table 2-1  
Dictionary Notation

Symbol	Meaning
A	The A register.
ADDRESS	Encompasses all the elements needed to specify an effective address. This term is used because various addressing types require you to specify the elements in different orders (such as indirect or pre- and post-indexing).
AP	Address pointer.
B	The 16-bit B register.
BR	Base register.
CB	Class bits.
CBIT	Bit 1 of the keys.
DAC	The double precision floating-point accumulator with 48 bits of mantissa and 16 bits of exponent.
Displacement	The number of halfwords to be added to the base register to form the effective address.
E	The 32-bit E register.
EA	Effective address.
F	Floating-point accumulator.
FAC	The single precision floating-point accumulator with 48 bits of mantissa and 16 bits of exponent.
FAR	Field address register.
FLR	Field length register.
Halfword	A 16-bit unit of memory.
I	Indirect bit.
L	The 32-bit L register.
LINK	Bit 3 of the keys. Not used in S and R modes.
Offset	The number of halfwords from the starting address of a segment.

Table 2-1 (continued)  
Dictionary Notation

Symbol	Meaning
QAC	The quad precision floating-point accumulator with 96 bits of mantissa and 16 bits of exponent.
skip	Skip next 16-bit halfword before continuing execution.
Word	A 32-bit unit of memory.
X	The X register (indexing).
XB	Auxiliary base register.
Y	The Y register (indexing).
m\ <i>n</i>	Specifies the number of bits, <i>n</i> , occupied by field <i>m</i> .
[ ]	Specifies an optional argument.

Resumable Instructions

Some assembly language instructions are resumable. When an interrupt is requested during the execution of an instruction, the processor usually services the interrupt at the end of execution before starting the next instruction. Some instructions, however, are too long or too complex for this to be desirable. When an interrupt is requested during one of these resumable instructions, the processor preserves the state of the interrupted instruction, handles the interrupt, then resumes the instruction at the point where the interrupt occurred. Table 2-2 lists the resumable assembly language instructions.

Table 2-2  
Resumable Instructions

Instructions			
ARGT	XAD	XBTD	XCM
XDTB	XDV	XED	XMP
XMV	ZCM	ZED	ZFIL
ZMV	ZMVD	ZTRN	STEX

These instructions depend on the settings in certain registers to determine whether they are being executed for the first or another time. In addition, some registers may be used for intermediate storage, modifying the previous contents as a side effect. Registers so modified are noted per instruction description.

### Storing Data Into the V and I Mode Instruction Stream

For the 6350 and 9750 to 9955 II, you must wait five instructions before executing data after any instruction that stores data into memory. If in doubt about the next five instructions (temporally) to be executed, use a mode change instruction to the current addressing mode, such as E64V, to allow the stored data to be executed. The rest of the 50 Series has no such restriction.

### Instruction Formats

All S, R, and V mode instructions belong to one of the following instruction types:

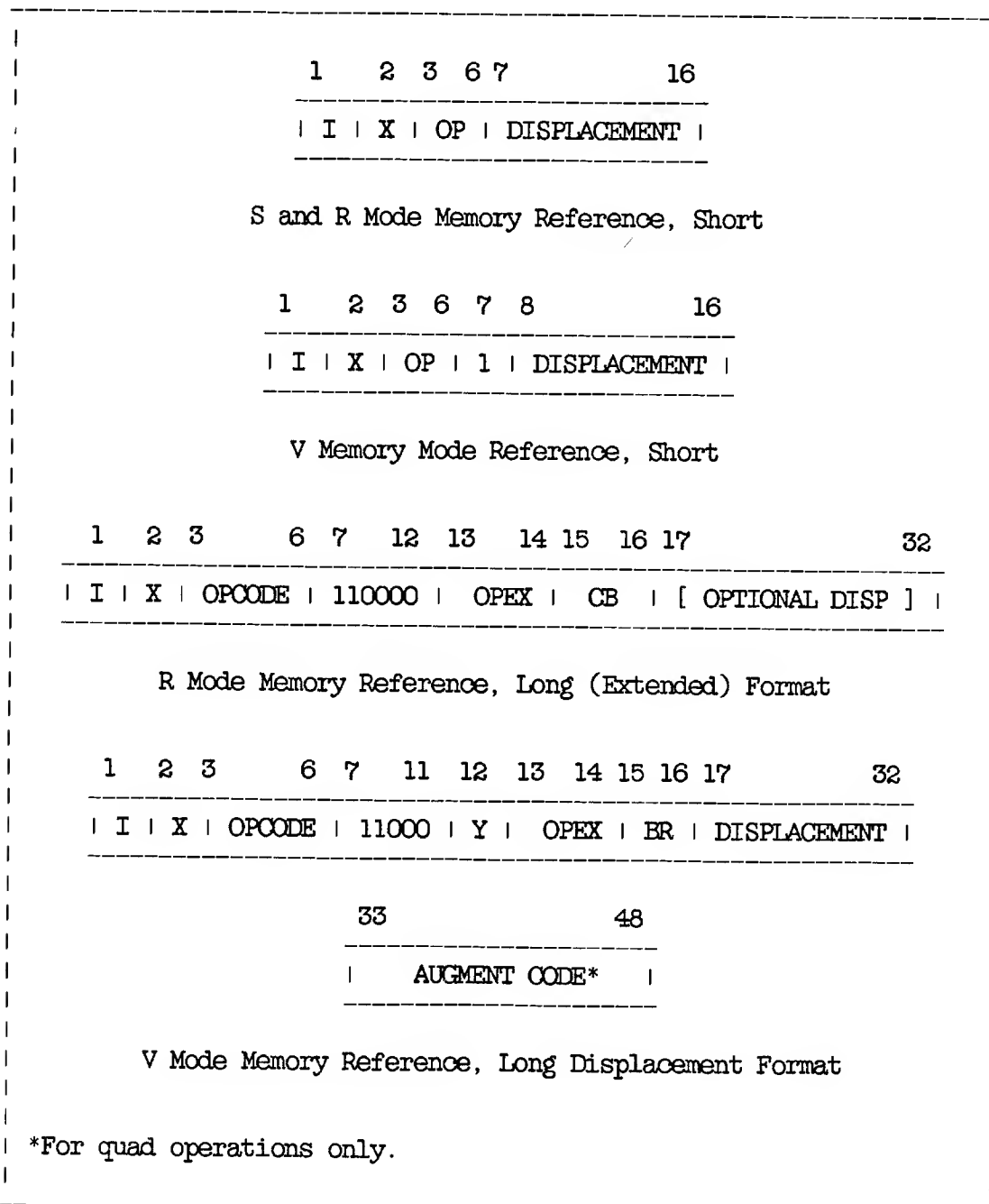
- S and R Mode Memory Reference, Short
- V Mode Memory Reference, Short
- R Mode Memory Reference, Long
- V Mode Memory Reference, Long
- V Mode Generic AP (Address Pointer)
- S, R, and V Mode Generic Type A
- S, R, and V Mode Generic Type B
- S, R, and V Mode Shift
- S, R, and V Mode Skip

The format of each instruction type is shown in Figure 2-1.

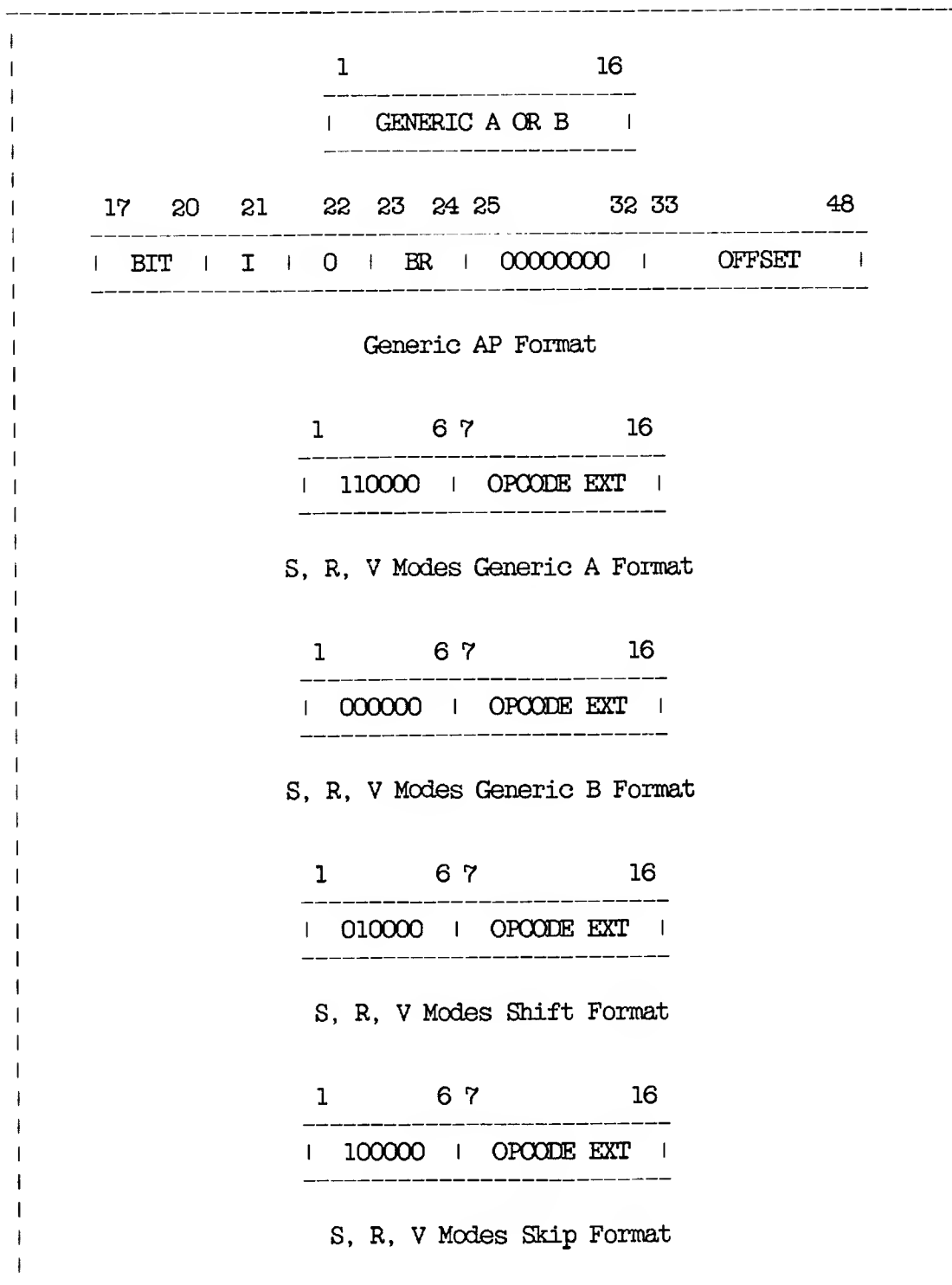
Short and long memory reference instructions have an opcode in bits 3 to 6. The value of this opcode ranges from 1 to '17, inclusive, with the exception of '14, which is reserved for I/O. For opcode '15, the X bit is part of the opcode.

In addition, long memory reference instructions have an opcode extension contained in bits 13 to 14. Generic AP instructions have a generic A or B format (where bits 7 to 16 contain the opcode extension) followed by a 32-bit address pointer.

Generic A and B, shift, and skip instructions are 16 bits long, all of which form an opcode. The values of bits 1 and 2 determine the basic instruction type: 11 for Generic A, 00 for Generic B, 01 for shifts, and 10 for skips. Bits 3 to 6 contain 0. Bits 7 to 16 contain an opcode extension. For shifts, bits 10 to 16 of the opcode extension contain the two's complement of the number of shifts to perform.



S, R, and V Mode Instruction Formats  
Figure 2-1



S, R, and V Mode Instruction Formats  
Figure 2-1 (continued)

INSTRUCTIONS

► A1A  
 Add 1 to A  
 1 1 0 0 0 0 1 0 1 0 0 0 0 1 1 0 (S, R, V mode form)

Adds 1 to the contents of A and stores the result in A. If A initially contains  $(2^{15})-1$ , an integer exception occurs and the instruction loads  $-(2^{15})$  into A. If no integer exception occurs, the instruction resets CBIT to 0. LINK contains the carry-out bit. The condition codes reflect the result of the operation. (See Appendix A.)

If an integer exception occurs and bit 8 of the keys contains a 0, the instruction sets CBIT to 1. If bit 8 contains a 1, the instruction sets CBIT to 1 and causes an integer exception fault. See Chapter 10 of the System Architecture Reference Guide for more information.

► A2A  
 Add 2 to A  
 1 1 0 0 0 0 0 0 1 1 0 0 0 1 0 0 (S, R, V mode form)

Adds 2 to the contents of A and stores the result in A. If A initially contains  $(2^{15})-1$  or  $(2^{15})-2$ , an integer exception occurs and the instruction loads  $-(2^{15})+1$  or  $-(2^{15})$ , respectively, into A. If no exception occurs, the instruction resets CBIT to 0. LINK contains the carry-out bit. The condition codes reflect the result of the operation. (See Appendix A.)

If an integer exception occurs and bit 8 of the keys contains a 0, the instruction sets CBIT to 1. If bit 8 contains a 1, the instruction sets CBIT to 1 and causes an integer exception fault. See Chapter 10 of the System Architecture Reference Guide for more information.

► ABQ address  
 Add Entry to Bottom of Queue  
 1 1 0 0 0 0 1 1 1 1 0 0 1 1 1 0 (V mode form)  
 AP\32

Adds the entry contained in A to the bottom of the queue referenced by the AP. (AP points to the queue's QCB.) Sets the condition codes to reflect EQ if the queue is full, or to NE if not full. Leaves the values of CBIT and LINK unchanged. See Chapters 6 and 11 of the System Architecture Reference Guide for more information about queues and queue operations.

## INSTRUCTION SETS GUIDE

► **ACA**  
Add CBIT to A  
1 1 0 0 0 0 1 0 1 0 0 0 1 1 1 0 (S, R, V mode form)

Adds the value of CBIT to the contents of A and stores the result in A. If the initial value of A is  $(2^{15})-1$  and CBIT is 1, the instruction loads  $-(2^{15})$  into A and an integer exception occurs. If no integer exception occurs, the instruction resets CBIT to 0. LINK contains the carry-out bit. The condition codes reflect the result of the operation. (See Appendix A.)

If an integer exception occurs and bit 8 of the keys contains a 0, the instruction sets CBIT to 1. If bit 8 contains a 1, the instruction sets CBIT to 1 and causes an integer exception fault. See Chapter 10 of the System Architecture Reference Guide for more information.

### Note

This instruction adds CBIT to bit 16 of A.

► **ADD address**  
Add  
I X 0 1 1 0 1 1 0 0 0 Y 0 0 BR\2 (V mode long)  
DISPLACEMENT\16  
  
I X 0 1 1 0 1 1 0 0 0 0 0 0 0 CB\2 (R mode long)  
[ DISPLACEMENT\16 ]  
  
I X 0 1 1 0 DISPLACEMENT\10 (S mode; R, V mode short)

Calculates an effective address, EA. Fetches the 16-bit contents of the location specified by EA and adds them to the contents of A. Stores the results in A.

If the resulting sum is less than or equal to  $(2^{15})-1$  and greater than or equal to  $-(2^{15})$ , the instruction resets CBIT to 0. If the sum is greater than or equal to  $2^{15}$ , an integer exception occurs. If the sum is less than or equal to  $-(2^{15})-1$ , an integer exception occurs.

When an integer exception occurs, the results are of the opposite sign of the correct answer. In addition, the 16 bits are the 16 LSBs of the correct answer, which needs 17 bits to be correctly represented.

If an integer exception occurs and bit 8 of the keys contains a 0, the instruction sets CBIT to 1. If bit 8 contains a 1, the instruction sets CBIT to 1 and causes an integer exception fault. See Chapter 10 of the System Architecture Reference Guide for more information.

At the end of the operation, LINK contains the carry-out bit. The condition codes reflect the result of the operation. (See Appendix A.)

► ADL address  
 Add Long  
 I X 0 1 1 0 1 1 0 0 0 Y 1 1 BR\2 (V mode form)  
 DISPLACEMENT\16

Calculates an effective address, EA. Fetches the 32-bit contents of the location specified by EA and adds them to the contents of L. Stores the results in L.

If the resulting sum is less than or equal to  $(2^{31})-1$  and greater than or equal to  $-(2^{31})$ , the instruction resets CBIT to 0. If the sum is greater than or equal to  $2^{31}$ , an integer exception occurs. If the sum is less than or equal to  $-(2^{31})-1$ , an integer exception occurs.

When an integer exception occurs, the results are of the opposite sign of the correct answer. In addition, the 32 bits are the 32 LSBs of the correct answer (that needs 33 bits to be correctly represented).

If an integer exception occurs and bit 8 of the keys contains a 0, the instruction sets CBIT to 1. If bit 8 contains a 1, the instruction sets CBIT to 1 and causes an integer exception fault. See Chapter 10 of the System Architecture Reference Guide for more information.

At the end of the operation, LINK contains the carry-out bit. The condition codes reflect the result of the operation. (See Appendix A.)

► ADLL  
 Add LINK to L  
 1 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 (V mode form)

Adds the contents of LINK to the contents of L and stores the result in L. If the initial value of L is  $(2^{31})-1$  and LINK is 1, an integer exception occurs. When an integer exception occurs, the results are of the opposite sign of the correct answer. In addition, the 32 bits are the 32 LSBs of the correct answer, which needs 33 bits to be correctly represented.

If no integer exception occurs, the instruction resets CBIT to 0. LINK contains the carry-out bit. The condition codes reflect the result of the operation. (See Appendix A.)

If an integer exception occurs and bit 8 of the keys contains a 0, the instruction sets CBIT to 1. If bit 8 contains a 1, the instruction sets CBIT to 1 and causes an integer exception fault. See Chapter 10 of the System Architecture Reference Guide for more information.

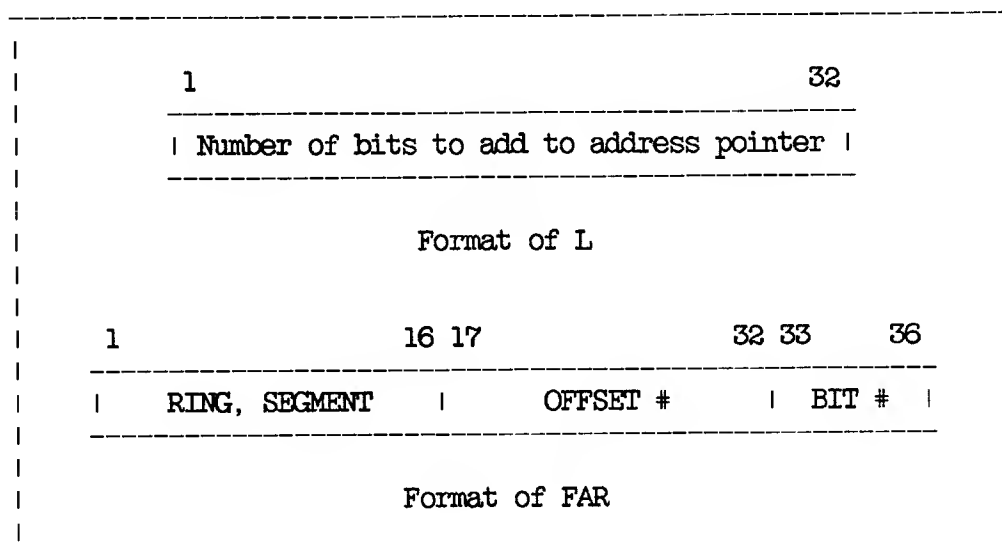
#### Note

This instruction adds the value of LINK to bit 32 of L.

► **ALFA far**  
 Add L to FAR  
 0 0 0 0 0 0 1 0 1 1 0 0 F 0 0 1 (V mode format)

Adds the two's complement value contained in L to the offset and bit number fields of FAR and stores the result in the specified FAR. Leaves the values of LINK and CBIT indeterminate. The values of the condition codes remain unchanged.

Figure 2-2 shows the format of L and the specified FAR for this instruction.



L and FAR Format for ALFA  
 Figure 2-2

► **ALL n**  
 A Left Logical  
 0 1 0 0 0 0 1 1 0 0 N\6 (S, R, V mode form)

Shifts the contents of A left the appropriate number of bits, bringing zeros in through bit 16 as needed. CBIT and LINK contain the value of the last bit shifted out; the values of the other bits shifted out are lost. Leaves the values of the condition codes unchanged. See Chapter 6 of the System Architecture Reference Guide for more information about shifts.

N contains the two's complement of the number of shifts to perform. If N contains 0, the instruction performs 64 shifts.

► ALR n  
 A Left Rotate  
 0 1 0 0 0 0 1 1 1 0 N\6 (S, R, V mode form)

Shifts the contents of A to the left, rotating bit 1 into bit 16. Stores the result in A. CBIT and LINK contain the value of the last bit rotated into bit 16. Leaves the values of the condition codes unchanged. See Chapter 6 of the System Architecture Reference Guide for more information about shifts.

N contains the two's complement of the number of shifts to perform. If N contains 0, the instruction performs 64 shifts.

► ALS n  
 A Arithmetic Left Shift  
 0 1 0 0 0 0 1 1 0 1 N\6 (S, R, V mode form)

Shifts the contents of A to the left, bringing zeros in on the right. Stores the result in A. If bit 1, the sign bit, changes state, the shift has resulted in a loss of significance and produces an integer exception. If no integer exception occurs, the instruction resets CBIT to 0. The value of LINK is indeterminate. Leaves the values of the condition codes unchanged. See Chapter 6 of the System Architecture Reference Guide for more information about shifts.

If an integer exception occurs and bit 8 of the keys contains 0, the instruction sets CBIT to 1. If bit 8 contains a 1, the instruction sets CBIT to 1 and causes an integer exception fault. See Chapter 10 of the System Architecture Reference Guide for more information.

► ANA address  
 AND to A  
 I X 0 0 1 1 1 1 0 0 0 Y 0 0 BR\2 (V mode long)  
 DISPLACEMENT\16  
  
 I X 0 0 1 1 1 1 0 0 0 0 0 0 CB\2 (R mode long)  
 [ DISPLACEMENT\16 ]  
  
 I X 0 0 1 1 DISPLACEMENT\10 (S mode; R, V mode short)

Calculates an effective address, EA. Logically ANDs the 16-bit contents of the location specified by EA with the contents of A, and stores the result in A. Leaves the values of CBIT, LINK, and the condition codes unchanged.

► ANL address  
 AND to A Long  
 I X 0 0 1 1 1 1 0 0 0 Y 1 1 BR\2 (V mode form)

Calculates a 32-bit effective address, EA. Logically ANDs the 32-bit contents of the location specified by EA with the contents of L, and stores the result in L. Leaves the values of CBIT, LINK, and the condition codes unchanged.

► ARGV  
 Argument Transfer  
 0 0 0 0 0 0 0 1 1 0 0 0 0 1 0 1 (V mode form)

Transfers arguments from a source procedure to a destination procedure. ARGV is fetched and executed only when the argument transfer phase of a procedure call (PCL) instruction is interrupted or faulted.

To perform a procedure call and argument transfer, the source procedure must contain the PCL instruction followed by a number of argument templates. The destination procedure must begin with the ARGV instruction. When the PCL instruction is executed, control transfers to the destination procedure, and the ARGV instruction uses the templates to form the actual arguments. The arguments are stored in the new stack frame as they are computed. At the end of the ARGV instruction, the values of CBIT, LINK, and the condition codes are indeterminate.

ARGV must be the first executable instruction in any destination procedure that will use arguments. For those procedures whose entry control blocks specify zero arguments, you must omit ARGV or you will destroy the return pointer for PCL, producing indeterminate results.

For more information about argument transfers, refer to the section on procedure calls in Chapter 8 of the System Architecture Reference Guide.

► ARL n  
 A Right Logical  
 0 1 0 0 0 0 0 1 0 0 N\6 (S, R, V mode form)

Shifts the contents of A right the appropriate number of bits, bringing zeros in through bit 1. CBIT and LINK contain the value of the last bit shifted out; the values of the other bits shifted out are lost. Leaves the values of the condition codes unchanged.

N contains the two's complement of the number of shifts to perform. If N contains 0, the instruction performs 64 shifts.

► ARR n  
 A Right Rotate  
 0 1 0 0 0 0 0 1 1 0 N\6 (S, R, V mode form)

Shifts the contents of A to the right, rotating bit 16 into bit 1. CBIT and LINK contain the value of the last bit rotated into bit 1. Leaves the values of the condition codes unchanged.

N contains the two's complement of the number of shifts to perform. If N contains 0, the instruction performs 64 shifts.

► ARS n  
 A Arithmetic Right Shift  
 0 1 0 0 0 0 0 1 0 1 N\6 (S, R, V mode form)

Shifts the contents of A to the right arithmetically, shifting copies of bit 1, the sign bit, into the vacated bits. CBIT and LINK contain the value of the last bit shifted out; the values of the other bits shifted out are lost. Leaves the values of the condition codes unchanged.

N contains the two's complement of the number of shifts to perform. If N contains 0, the instruction performs 64 shifts.

► ATQ address  
 Add Entry to Top of Queue  
 1 1 0 0 0 0 1 1 1 0 0 1 1 1 1 (V mode form)  
 AP\32

Adds the entry contained in A to the top of the queue referenced by the AP. (AP points to the queue's QCB.) Sets the condition codes to reflect EQ if the queue is full, or to NE if not full. Leaves the values of CBIT and LINK unchanged. For more information about queues and queue manipulation, see Chapters 6 and 11 of the System Architecture Reference Guide.

► BCEQ address  
 Branch on Condition Code EQ  
 1 1 0 0 0 0 1 1 1 0 0 0 0 0 1 0 (V mode form)  
 ADDRESS\16

If the condition codes reflect equal to 0, the instruction loads the specified address into the program counter. This address must be within the current segment. If the condition codes reflect some other condition, execution continues with the next instruction. Leaves the values of CBIT, LINK, and the condition codes unchanged.

► BGE address  
 Branch on Condition Code GE  
 1 1 0 0 0 0 1 1 1 0 0 0 0 0 1 0 1 (V mode form)  
 ADDRESS\16

If the condition codes reflect greater than or equal to 0, the instruction loads the specified address into the program counter. This address must be within the current segment. If the condition codes reflect some other condition, execution continues with the next instruction. Leaves the values of CBIT, LINK, and the condition codes unchanged.

► BGT address  
 Branch on Condition Code GT  
 1 1 0 0 0 0 1 1 1 0 0 0 0 0 0 1 (V mode form)  
 ADDRESS\16

If the condition codes reflect greater than 0, the instruction loads the specified address into the program counter. This address must be within the current segment. If the condition codes reflect some other condition, execution continues with the next instruction. Leaves the values of CBIT, LINK, and the condition codes unchanged.

► BCLE address  
 Branch on Condition Code LE  
 1 1 0 0 0 0 1 1 1 0 0 0 0 0 0 0 (V mode form)  
 ADDRESS\16

If the condition codes reflect less than or equal to 0, the instruction loads the specified address into the program counter. This address must be within the current segment. If the condition codes reflect some other condition, execution continues with the next instruction. Leaves the values of CBIT, LINK, and the condition codes unchanged.

► BCLT address  
 Branch on Condition Code LT  
 1 1 0 0 0 0 1 1 1 0 0 0 0 1 0 0 (V mode form)  
 ADDRESS\16

If the condition codes reflect less than 0, the instruction loads the specified address into the program counter. This address must be within the current segment. If the condition codes reflect some other condition, execution continues with the next instruction. Leaves the values of CBIT, LINK, and the condition codes unchanged.

► BCNE address  
 Branch on Condition Code NE  
 1 1 0 0 0 0 1 1 1 0 0 0 0 0 1 1 (V mode form)  
 ADDRESS\16

If the condition codes reflect not equal to 0, the instruction loads the specified address into the program counter. This address must be within the current segment. If the condition codes reflect some other condition, execution continues with the next instruction. Leaves the values of CBIT, LINK, and the condition codes unchanged.

► BCR address  
 Branch on CBIT Reset to 0  
 1 1 0 0 0 0 1 1 1 1 0 0 0 1 0 1 (V mode form)  
 ADDRESS\16

If CBIT has the value 0, the instruction loads the specified address into the program counter. This address must be within the current segment. If CBIT has the value 1, execution continues with the next instruction. Leaves the values of CBIT, LINK, and the condition codes unchanged.

► BCS address  
 Branch on CBIT Set to 1  
 1 1 0 0 0 0 1 1 1 1 0 0 0 1 0 0 (V mode form)  
 ADDRESS\16

If CBIT has the value 1, the instruction loads the specified address into the program counter. This address must be within the current segment. If CBIT has the value 0, execution continues with the next instruction. Leaves the values of CBIT, LINK, and the condition codes unchanged.

► **BDX address**  
 Branch on Decremented X  
 1 1 0 0 0 0 0 1 1 1 0 1 1 1 0 0 (V mode form)  
 ADDRESS\16

Decrements the contents of X by one and stores the result in X. If the decremented value is not equal to 0, loads the specified address into the program counter. This address must be within the current segment. If the decremented value is equal to 0, execution continues with the next instruction. Leaves the values of CBIT, LINK, and the condition codes unchanged.

► **BDY address**  
 Branch on Decremented Y  
 1 1 0 0 0 0 0 1 1 1 0 1 0 1 0 0 (V mode form)  
 ADDRESS\16

Decrements the contents of Y by one and stores the result in Y. If the decremented value is not equal to 0, loads the specified address into the program counter. This address must be within the current segment. If the decremented value is equal to 0, execution continues with the next instruction. Leaves the values of CBIT, LINK, and the condition codes unchanged.

► **BEQ address**  
 Branch on A Equal to 0  
 1 1 0 0 0 0 0 1 1 0 0 0 1 0 1 0 (V mode form)  
 ADDRESS\16

If the contents of A are equal to 0, the instruction loads the specified address into the program counter. This address must be within the current segment. If the A contents are not equal to 0, execution continues with the next instruction. The condition codes contain the result of the comparison. (See Appendix A.) Leaves the values of LINK and CBIT unchanged.

► **BFEQ address**  
 Branch on Floating Accumulator Equal to 0  
 1 1 0 0 0 0 1 1 1 0 0 0 1 0 1 0 (V mode form)  
 ADDRESS\16

If the contents of the floating accumulator are equal to 0, the instruction loads the specified address into the program counter. This address must be within the current segment. If the floating accumulator contents are not equal to 0, execution continues with the next instruction. The condition codes contain the result of the comparison. (See Appendix A.) Leaves the values of LINK and CBIT unchanged.

BFEQ works correctly only on normalized or nearly normalized numbers, because it checks the first 32 fraction bits only for equal to zero and less than zero. (See Chapter 6 in the System Architecture Reference Guide.)

► BFEQ address  
 Branch on Floating Accumulator Greater Than or Equal to 0  
 1 1 0 0 0 0 1 1 1 0 0 0 1 1 0 1 (V mode form)  
 ADDRESS\16

If the contents of the floating accumulator are greater than or equal to 0, the instruction loads the specified address into the program counter. This address must be within the current segment. If the floating accumulator contents are less than 0, execution continues with the next instruction. The condition codes contain the result of the comparison. (See Appendix A.) Leaves the values of LINK and CBIT unchanged. BFEQ works correctly only on normalized or nearly normalized numbers, because it checks the first 32 fraction bits only for equal to zero and less than zero. (See Chapter 6 in the System Architecture Reference Guide.)

► BFGT address  
 Branch on Floating Accumulator Greater Than 0  
 1 1 0 0 0 0 1 1 1 0 0 0 1 0 0 1 (V mode form)  
 ADDRESS\16

If the contents of the floating accumulator are greater than 0, the instruction loads the specified address into the program counter. This address must be within the current segment. If the floating accumulator contents are less than or equal to 0, execution continues with the next instruction. The condition codes contain the result of the comparison. (See Appendix A.) Leaves the values of LINK and CBIT unchanged. BFGT works correctly only on normalized or nearly normalized numbers, because it checks the first 32 fraction bits only for equal to zero and less than zero. (See Chapter 6 in the System Architecture Reference Guide.)

► BFLE address  
 Branch on Floating Accumulator Less Than or Equal to 0  
 1 1 0 0 0 0 1 1 1 0 0 0 1 0 0 0 (V mode form)  
 ADDRESS\16

If the floating accumulator contents are less than or equal to 0, BFLE loads the specified address into the program counter. This address must be within the current segment. If the floating accumulator contents are greater than 0, execution continues with the next instruction. The condition codes contain the comparison result. (See Appendix A.) Leaves the values of LINK and CBIT unchanged.

BFLTE works correctly only on normalized or nearly normalized numbers, because it checks the first 32 fraction bits only for equal to zero and less than zero. (See Chapter 6 in the System Architecture Reference Guide.)

► BFLT address  
 Branch on Floating Accumulator Less Than 0  
 1 1 0 0 0 0 1 1 1 0 0 0 1 1 0 0 (V mode form)  
 ADDRESS\16

If the contents of the floating accumulator are less than 0, the instruction loads the specified address into the program counter. This address must be within the current segment. If the floating accumulator contents are greater than or equal to 0, execution continues with the next instruction. The condition codes contain the result of the comparison. (See Appendix A.) Leaves the values of LINK and CBIT unchanged. BFLT works correctly only on normalized or nearly normalized numbers, because it checks the first 32 fraction bits only for equal to zero and less than zero. (See Chapter 6 in the System Architecture Reference Guide.)

► BFNE address  
 Branch on Floating Accumulator Not Equal to 0  
 1 1 0 0 0 0 1 1 1 0 0 0 1 0 1 1 (V mode form)  
 ADDRESS\16

If the contents of the floating accumulator are not equal to 0, the instruction loads the specified address into the program counter. This address must be within the current segment. If the floating accumulator contents are equal to 0, execution continues with the next instruction. The condition codes contain the result of the comparison. (See Appendix A.) Leaves the values of LINK and CBIT unchanged. BFNE works correctly only on normalized or nearly normalized numbers because it checks the first 32 fraction bits only for equal to zero and less than zero. (See Chapter 6 in the System Architecture Reference Guide.)

► BGE address  
 Branch on A Greater Than or Equal to 0  
 1 1 0 0 0 0 0 1 1 0 0 0 1 1 0 1 (V mode form)  
 ADDRESS\16

If the contents of A are greater than or equal to 0, the instruction loads the specified address into the program counter. This address must be within the current segment. If the A contents are less than 0, execution continues with the next instruction. The condition codes contain the result of the comparison. (See Appendix A.) Leaves the values of LINK and CBIT unchanged. This instruction has the same operation as BLGE.

► BGT address  
 Branch on A Greater Than 0  
 1 1 0 0 0 0 0 1 1 0 0 0 1 0 0 1 (V mode form)  
 ADDRESS\16

If the contents of A are greater than 0, the instruction loads the specified address into the program counter. This address must be within the current segment. If the A contents are less than or equal to 0, execution continues with the next instruction. The condition codes contain the result of the comparison. (See Appendix A.) Leaves the values of LINK and CBIT unchanged.

► BIX address  
 Branch on Incremented X  
 1 1 0 0 0 0 1 0 1 1 0 1 1 1 0 0 (V mode form)  
 ADDRESS\16

Increments the contents of X by one and stores the result in X. If the incremented value is not equal to 0, loads the specified address into the program counter. This address must be within the current segment. If the incremented value is equal to 0, execution continues with the next instruction. Leaves the values of CBIT, LINK, and the condition codes unchanged.

► BITY address  
 Branch on Incremented Y  
 1 1 0 0 0 0 1 0 1 1 0 1 0 1 0 0 (V mode form)  
 ADDRESS\16

Increments the contents of Y by one and stores the result in Y. If the incremented value is not equal to 0, loads the specified address into the program counter. This address must be within the current segment. If the incremented value is equal to 0, execution continues with the next instruction. Leaves the values of CBIT, LINK, and the condition codes unchanged.

► BLE address  
 Branch on A Less Than or Equal to 0  
 1 1 0 0 0 0 0 1 1 0 0 0 1 0 0 0 (V mode form)  
 ADDRESS\16

If the contents of A are less than or equal to 0, the instruction loads the specified address into the program counter. This address must be within the current segment. If the A contents are greater than 0, execution continues with the next instruction. The condition codes contain the result of the comparison. (See Appendix A.) Leaves the values of LINK and CBIT unchanged.

► **BLEQ address**  
 Branch on L Equal to 0  
 1 1 0 0 0 0 0 1 1 1 0 0 0 0 1 0 (V mode form)  
 ADDRESS\16

If the contents of L are equal to 0, the instruction loads the specified address into the program counter. This address must be within the current segment. If the L contents are not equal to 0, execution continues with the next instruction. The condition codes contain the result of the comparison. (See Appendix A.) Leaves the values of LINK and CBIT unchanged.

► **BLGE address**  
 Branch on L Greater Than or Equal to 0  
 1 1 0 0 0 0 0 1 1 0 0 0 1 1 0 1 (V mode form)  
 ADDRESS\16

If the contents of L are greater than or equal to 0, the instruction loads the specified address into the program counter. This address must be within the current segment. If the L contents are less than 0, execution continues with the next instruction. The condition codes contain the result of the comparison. (See Appendix A.) Leaves the values of LINK and CBIT unchanged. This instruction has the same operation as BGE.

► **BLGT address**  
 Branch on L Greater Than 0  
 1 1 0 0 0 0 0 1 1 1 0 0 0 0 0 1 (V mode form)  
 ADDRESS\16

If the contents of L are greater than 0, the instruction loads the specified address into the program counter. This address must be within the current segment. If the L contents are less than or equal to 0, execution continues with the next instruction. The condition codes contain the result of the comparison. (See Appendix A.) Leaves the values of LINK and CBIT unchanged.

► **BLLE address**  
 Branch on L Less Than or Equal to 0  
 1 1 0 0 0 0 0 1 1 1 0 0 0 0 0 0 (V mode form)  
 ADDRESS\16

If the contents of L are less than or equal to 0, the instruction loads the specified address into the program counter. This address must be within the current segment. If the L contents are greater than 0, execution continues with the next instruction. The condition codes contain the result of the comparison. (See Appendix A.) Leaves the values of LINK and CBIT unchanged.

► **BLLT address**  
 Branch on L Less Than 0  
 1 1 0 0 0 0 0 1 1 0 0 0 1 1 0 0 (V mode form)  
 ADDRESS\16

If the contents of L are less than 0, the instruction loads the specified address into the program counter. This address must be within the current segment. If the L contents are greater than or equal to 0, execution continues with the next instruction. The condition codes contain the result of the comparison. (See Appendix A.) Leaves the values of LINK and CBIT unchanged. This instruction has the same operation as BLT.

► **BLNE address**  
 Branch on L Not Equal to 0  
 1 1 0 0 0 0 0 1 1 1 0 0 0 0 1 1 (V mode form)  
 ADDRESS\16

If the contents of L are not equal to 0, the instruction loads the specified address into the program counter. This address must be within the current segment. If the L contents are equal to 0, execution continues with the next instruction. The condition codes contain the result of the comparison. (See Appendix A.) Leaves the values of LINK and CBIT unchanged.

► **BLR address**  
 Branch on LINK Reset to 0  
 1 1 0 0 0 0 1 1 1 1 0 0 0 1 1 1 (V mode form)  
 ADDRESS\16

If LINK has the value 0, the instruction loads the specified address into the program counter. This address must be within the current segment. If LINK has the value 1, execution continues with the next instruction. Leaves the values of CBIT, LINK, and the condition codes unchanged.

► **BLS address**  
 Branch on LINK Set to 1  
 1 1 0 0 0 0 1 1 1 1 0 0 0 1 1 0 (V mode form)  
 ADDRESS\16

If LINK has the value 1, the instruction loads the specified address into the program counter. This address must be within the current segment. If LINK has the value 0, execution continues with the next instruction. Leaves the values of CBIT, LINK, and the condition codes unchanged.

► **BLT address**  
 Branch on A Less Than 0  
 1 1 0 0 0 0 0 1 1 0 0 0 1 1 0 0 (V mode form)  
 ADDRESS\16

If the contents of A are less than 0, the instruction loads the specified address into the program counter. This address must be within the current segment. If the A contents are greater than or equal to 0, execution continues with the next instruction. The condition codes contain the result of the comparison. (See Appendix A.) Leaves the values of LINK and CBIT unchanged. This instruction has the same operation as BLLT.

► **BMEQ address**  
 Branch on Magnitude Condition EQ  
 1 1 0 0 0 0 1 1 1 0 0 0 0 0 1 0 (V mode form)  
 ADDRESS\16

If the condition codes indicate magnitude equal to 0, the instruction loads the specified address into the program counter, like BCEQ. BMEQ is intended for magnitude comparisons after a compare or subtract instruction. This address must be within the current segment. If the condition codes indicate some other condition, execution continues with the next instruction. Leaves the values of CBIT, LINK, and the condition codes unchanged.

► **BMGE address**  
 Branch on Magnitude Condition GE  
 1 1 0 0 0 0 1 1 1 1 0 0 0 1 1 0 (V mode form)  
 ADDRESS\16

If LINK has the value 1, the instruction loads the specified address into the program counter, like BLS. BMGE is used to determine if the magnitude of the A/L register quantity was greater than or equal to the memory quantity after a compare or subtract instruction. This address must be within the current segment. If LINK has the value 0, execution continues with the next instruction. Leaves the values of CBIT, LINK, and the condition codes unchanged.

► **BMGT address**  
 Branch on Magnitude Condition GT  
 1 1 0 0 0 0 1 1 1 1 0 0 1 0 0 0 (V mode form)  
 ADDRESS\16

If LINK is 1 and the condition codes reflect not equal to 0, the instruction loads the specified address into the program counter. This address must be within the current segment. If some other condition exists, execution continues with the next instruction. Leaves the values of CBIT, LINK, and the condition codes unchanged.

► **BMLE address**  
 Branch on Magnitude Condition LE  
 1 1 0 0 0 0 1 1 1 1 0 0 1 0 0 1 (V mode form)  
 ADDRESS\16

If LINK is 0 or the condition codes reflect equal to 0, the instruction loads the specified address into the program counter. This address must be within the current segment. If some other condition exists, execution continues with the next instruction. Leaves the values of CBIT, LINK, and the condition codes unchanged.

► **BMLT address**  
 Branch on Magnitude Condition LT  
 1 1 0 0 0 0 1 1 1 1 0 0 0 1 1 1 (V mode form)  
 ADDRESS\16

If LINK has the value 0, the instruction loads the specified address into the program counter, like BLR. BMLT is used to determine if the magnitude of the A/L register quantity is less than the memory quantity after a compare or subtract instruction. This address must be within the current segment. If LINK has the value 1, execution continues with the next instruction. Leaves the values of LINK, CBIT, and the condition codes unchanged.

► **BMNE address**  
 Branch on Magnitude Condition NE  
 1 1 0 0 0 0 1 1 1 0 0 0 0 0 1 1 (V mode form)  
 ADDRESS\16

If the condition codes indicate magnitude not equal to 0, the instruction loads the specified address into the program counter, like BCNE. BMNE is intended for magnitude comparisons after a compare or subtract instruction. This address must be within the current segment. If the condition codes reflect some other condition, execution continues with the next instruction. Leaves the values of CBIT, LINK, and the condition codes unchanged.

► BNE address  
 Branch on A Not Equal to 0  
 1 1 0 0 0 0 0 1 1 0 0 0 1 0 1 1 (V mode form)  
 ADDRESS\16

If the contents of A are not equal to 0, the instruction loads the specified address into the program counter. This address must be within the current segment. If the A contents are equal to 0, execution continues with the next instruction. The condition codes contain the result of the comparison. (See Appendix A.) Leaves the values of LINK and CBIT unchanged.

► CAL  
Clear A Left Byte  
1 1 0 0 0 0 1 0 0 0 1 0 1 0 0 0 (S, R, V mode form)

Clears the left byte of A to 0. Leaves the values of CBIT, LINK, and the condition codes unchanged.

► CALF address  
Call Fault Handler  
0 0 0 0 0 0 0 1 1 1 0 0 0 1 0 1 (V mode form)  
AP\32

The address pointer in this instruction is to the ECB of a fault routine. The instruction uses this pointer to transfer control to the fault routine as if the transfer were a normal procedure call with no arguments passed. The values of CBIT, LINK, and the condition codes are indeterminate. See Chapter 10 of the System Architecture Reference Guide for more information.

► CAR  
Clear A Right Byte  
1 1 0 0 0 0 1 0 0 0 1 0 0 1 0 0 (S, R, V mode form)

Clears the right byte of A to 0. Leaves the values of CBIT, LINK, and the condition codes unchanged.

► CAS address  
Compare A and Skip  
I X 1 0 0 1 1 1 0 0 0 Y 0 0 BR\2 (V mode long)  
DISPLACEMENT\16

I X 1 0 0 1 1 1 0 0 0 0 0 0 0 CB\2 (R mode long)  
[ DISPLACEMENT\16 ]

I X 1 0 0 1 DISPLACEMENT\10 (S mode; R, V mode short)

Calculates an effective address, EA. For 16-bit two's complement signed values only, compares the contents of the A register to the contents of the location specified by EA and skips as follows:

<u>Condition</u>	<u>Skip</u>
Contents of A > contents of EA.	No skip.
Contents of A = contents of EA.	Skip 16 bits (one halfword).
Contents of A < contents of EA.	Skip 32 bits (two halfwords).

The value of CBIT is unchanged. LINK contains the carry-out bit. The condition codes reflect the result of the operation. (See Appendix A.)

► CAZ

Compare A With 0

1 1 0 0 0 0 0 0 1 0 0 0 1 1 0 0 (S, R, V mode form)

Compares the contents of A with 0. Skips as follows:

<u>Condition</u>	<u>Skip</u>
Contents of A > 0.	No skip.
Contents of A = 0.	Skip 16 bits (one halfword).
Contents of A < 0.	Skip 32 bits (two halfwords).

The value of CBIT is unchanged. LINK contains the carry-out bit. The condition codes reflect the result of the operation. (See Appendix A.)

► CEA

Compute Effective Address

0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 1 (S, R mode form)

Interprets the contents of A as a 16-bit indirect address in the current addressing mode. Calculates an effective address, EA, from the indirect address and loads the final address into A. Leaves the values of CBIT, LINK, and the condition codes unchanged.

► OGT

Computed GOTO

0 0 0 0 0 0 0 1 0 1 1 0 0 1 1 0 0 (V mode form)

INTEGER N\16

BRANCH ADDRESS 1\16

..

BRANCH ADDRESS (N-1)\16

If the contents of A are greater than or equal to 1 and less than the specified integer N that follows the opcode, the instruction adds the contents of A to the contents of the program counter to form an address. (The program counter points to the integer N following the opcode.) Loads the contents of the location specified by this address into the program counter. If the contents of A are not within this range, the instruction adds integer N to the contents of the program counter and stores the result in the program counter. The values of CBIT, LINK, and the condition codes are indeterminate.

Note

Each of the branch addresses following the OGT instruction specifies a location within the current procedure segment.

► CHS  
Change Sign  
1 1 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 (S, R, V mode form)

Complements bit 1 of A. Leaves the values of CBIT, LINK, and the condition codes unchanged.

► CLS address  
Compare L and Skip  
I X 1 0 0 1 1 1 0 0 0 Y 1 1 BR\2 (V mode form)  
DISPLACEMENT\16

Calculates an effective address, EA. For 32-bit two's complement signed values only, compares the contents of L to the contents of the 32-bit location specified by EA and skips as follows.

<u>Condition</u>	<u>Skip</u>
Contents of L > contents of EA.	No skip.
Contents of L = contents of EA.	Skip 16 bits (one halfword).
Contents of L < contents of EA.	Skip 32 bits (two halfwords).

The value of CBIT is unchanged. LINK contains the carry-out bit. The condition codes reflect the result of the operation. (See Appendix A.)

► CMA  
Complement A  
1 1 0 0 0 0 0 1 0 0 0 0 0 0 0 1 (S, R, V mode form)

Forms the one's complement of the contents of A by inverting the value of each bit, and stores the result in A. Leaves the values of CBIT, LINK, and the condition codes unchanged.

► **CRA**  
 Clear A to 0  
 1 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 (S, R, V mode form)

Clears the contents of A to 0. Leaves the values of CBIT, LINK, and the condition codes unchanged.

► **CRB**  
 Clear B to 0  
 1 1 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 (S, R, V mode form)  
 1 1 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0

Clears the contents of B to 0. Leaves the values of CBIT, LINK, and the condition codes unchanged.

#### Note

Opcode '140014 executes both a CRB and a FDBL. This is a conversion aid for P300 programs. This opcode should not be used; it is implemented for compatibility's sake only.

► **CRE**  
 Clear E to 0  
 1 1 0 0 0 0 1 1 0 0 0 0 0 1 0 0 (V mode form)

Clears the contents of E to 0. Leaves the values of CBIT, LINK, and the condition codes unchanged.

► **CRL**  
 Clear L to 0  
 1 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 (S, R, V mode form)

Clears the contents of L to 0. Leaves the values of CBIT, LINK, and the condition codes unchanged.

► **CRLE**  
 Clear L and E to 0  
 1 1 0 0 0 0 1 1 0 0 0 0 0 1 0 0 0 (V mode form)

Clears the contents of E and L to 0. Leaves the values of LINK, CBIT, and the condition codes unchanged.

► CSA  
Copy Sign of A  
1 1 0 0 0 0 0 0 1 1 0 1 0 0 0 0 (S, R, V mode form)

Sets CBIT equal to the value of bit 1 of A and clears bit 1 of A to 0.  
The value of LINK is indeterminate. Leaves the values of the condition codes unchanged.

► DAD address  
 Double Add  
 I X 0 1 1 0 1 1 0 0 0 0 0 0 CB\2 (R mode long)  
 [ DISPLACEMENT\16 ]

I X 0 1 1 0 DISPLACEMENT\10 (S, R mode form)

Calculates an effective address, EA. Fetches the 31-bit contents of the location specified by EA and adds them to the 31-bit contents of A and B. Stores the result in A and B.

If the result is greater than or equal to  $2^{30}$ , an integer exception occurs and the instruction loads bit 1 of A with a 1, and bits 2 to 16 of A and bits 2 to 16 of B with  $(\text{result} - (2^{30}))$ . Bit 1 of B contains 0.

If the result is less than  $-(2^{30})$ , an integer exception occurs and the instruction loads bit 1 of A with a 0 and bits 2 to 16 of A and bits 2 to 16 of B with the negative of  $(\text{result} + (2^{30}))$ . Bit 1 of B contains 0.

If no integer exception occurs, CBIT is reset to 0. At the end of the instruction, LINK contains the carry-out bit. The condition codes reflect the result of the operation. (See Appendix A.)

If an integer exception occurs and bit 8 of the keys contains a 0, the instruction sets CBIT to 1. If bit 8 contains a 1, the instruction sets CBIT to 1 and causes an integer exception fault. See Chapter 10 of the System Architecture Reference Guide for more information.

#### Notes

1. Bit 17 of each 31-bit integer must be 0. If nonzero, unpredictable results will occur.
2. This instruction executes in double precision mode only.

► DEL  
 Enter Double Precision Mode  
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 (S, R mode form)

Enters double precision mode by setting bit 2 of the keys to 1. Subsequent LDA, STA, ADD, and SUB instructions manipulate 31-bit integers and are interpreted as DLD, DST, DAD, and DSB, respectively. Leaves the values of CBIT, LINK, and the condition codes unchanged. In V or I mode, bit 2 of the keys has no effect.

► DFAD address  
 Double Precision Floating Add  
 I X 0 1 1 0 1 1 0 0 0 Y 1 0 BR\2 (V mode long)  
 DISPLACEMENT\16  
  
 I X 0 1 1 0 1 1 0 0 0 0 1 0 CB\2 (R mode long)  
 [ DISPLACEMENT\16 ]

Calculates an effective address, EA. Adds the double precision number in the location specified by EA to the 64-bit contents of the DAC. (See Chapter 6 of the System Architecture Reference Guide for more information.) Normalizes the result and loads it into the DAC. An overflow causes a floating-point exception. If no floating-point exception occurs, CBIT is reset to 0. The values of LINK and the condition codes are indeterminate.

For 750 and 850 processors, exponent underflow is detected, but exponent overflow is not.

If a floating-point exception occurs and bit 7 of the keys contains a 1, the instruction sets CBIT to 1. If bit 7 contains a 0, the instruction sets CBIT to 1 and causes a floating-point exception fault. See Chapter 10 of the System Architecture Reference Guide for more information.

► DFCM  
 Double Precision Floating Complement  
 1 1 0 0 0 0 0 1 0 1 1 1 1 1 0 0 (R, V mode form)

Forms the two's complement of the double precision number in the DAC and normalizes it if necessary. (See Chapter 6 of the System Architecture Reference Guide.) Stores the result in the DAC. An overflow causes a floating-point exception. If no floating-point exception occurs, CBIT is reset to 0. The values of LINK and the condition codes are indeterminate.

If a floating-point exception occurs and bit 7 of the keys contains a 1, the instruction sets CBIT to 1. If bit 7 contains a 0, the instruction sets CBIT to 1 and causes a floating-point exception fault. See Chapter 10 of the System Architecture Reference Guide for more information.

► DFCS address  
 Double Precision Floating Point Compare and Skip  
 I X 1 0 0 1 1 1 0 0 0 Y 1 0 BR\2 (V mode long)  
 DISPLACEMENT\16

I X 1 0 0 1 1 1 0 0 0 0 1 0 CB\2 (R mode long)  
 [ DISPLACEMENT\16 ]

Calculates an effective address, EA. Compares the DAC contents (see Chapter 6 of the System Architecture Reference Guide) to the contents of the 64-bit location specified by EA and skips as follows.

<u>Condition</u>	<u>Skip</u>
DAC contents > EA contents.	No skip.
DAC contents = EA contents.	Skip 16 bits (one halfword).
DAC contents < EA contents.	Skip 32 bits (two halfwords).

The values of CBIT, LINK, and the condition codes are indeterminate. On some processors, DFCS works correctly only on normalized numbers as follows. The comparison has a maximum of three sequential stages: first the signs, then the exponents, and finally the fractions of the two numbers are compared for equality. If the comparison during any one of these stages reveals an inequality, the results are returned and the instruction ends. Unnormalized numbers are unexpected and produce unexpected results. Other processors actually perform a subtract operation, resulting in a proper comparison.

► DFDV address  
 Double Precision Floating Point Divide  
 I X 1 1 1 1 1 1 0 0 0 Y 1 0 BR\2 (V mode long)  
 DISPLACEMENT\16

I X 1 1 1 1 1 1 0 0 0 0 1 0 CB\2 (R mode long)  
 [ DISPLACEMENT\16 ]

Calculates an effective address, EA. Divides the contents of the DAC by the contents of the location specified by EA. (See Chapter 6 of the System Architecture Reference Guide.) Normalizes the result and stores the whole quotient in the DAC. An overflow or a divide by 0 causes a floating-point exception. If no floating-point exception occurs, CBIT is reset to 0. The values of LINK and the condition codes are indeterminate.

If a floating-point exception occurs and bit 7 of the keys contains a 1, the instruction sets CBIT to 1. If bit 7 contains a 0, the instruction sets CBIT to 1 and causes a floating-point exception fault. See Chapter 10 of the System Architecture Reference Guide.

► DFLD address  
 Double Precision Floating Point Load  
 I X 0 0 1 0 1 1 0 0 0 Y 1 0 BR\2 (V mode long form)  
 DISPLACEMENT\16

I X 0 0 1 0 1 1 0 0 0 0 1 0 CB\2 (R mode long form)  
 [ DISPLACEMENT\16 ]

Calculates an effective address, EA. Loads the 64-bit contents of the location specified by EA into the DAC. (See Chapter 6 of the System Architecture Reference Guide.) Leaves the values of LINK, CBIT, and the condition codes unchanged.

#### Note

This instruction does not normalize the result before loading it into the DAC.

► DFLX address  
 Double Precision Floating Point Load Index  
 I 0 1 1 0 1 1 1 0 0 0 Y 1 0 BR\2 (V mode long)  
 DISPLACEMENT\16

I 0 1 1 0 1 1 1 0 0 0 0 1 0 CB\2 (R mode long)  
 [ DISPLACEMENT\16 ]

Calculates an effective address, EA. Loads the index register, X, with four times the 16-bit contents of the location specified by EA. Leaves the values of CBIT, LINK, and the condition codes unchanged.

#### Note

DFLX cannot do indexing. See Appendix B for more information.

► DFMP address  
 Double Precision Floating Point Multiply  
 I X 1 1 1 0 1 1 0 0 0 Y 1 0 BR\2 (V mode long)  
 DISPLACEMENT\16

I X 1 1 1 0 1 1 0 0 0 0 1 0 CB\2 (R mode long)  
 [ DISPLACEMENT\16 ]

Calculates an effective address, EA. Multiplies the contents of the DAC by the 64-bit contents of the location specified by EA. (See Chapter 6 of the System Architecture Reference Guide.) Normalizes the result, if necessary, and stores it in the DAC. An overflow causes a floating-point exception; if none occurs, CBIT is reset to 0. The values of LINK and the condition codes are indeterminate.

If a floating-point exception occurs and bit 7 of the keys contains a 1, the DFMP instruction sets CBIT to 1. If bit 7 contains a 0, the instruction sets CBIT to 1 and causes a floating-point exception fault. See Chapter 10 of the System Architecture Reference Guide for more information.

► DFSP address  
 Double Precision Floating Point Subtract  
 I X 0 1 1 1 1 0 0 0 Y 1 0 BR\2 (V mode long)  
 DISPLACEMENT\16  
  
 I X 0 1 1 1 1 0 0 0 0 1 0 CB\2 (R mode long)  
 [ DISPLACEMENT\16 ]

Calculates an effective address, EA. Subtracts the 64-bit contents of the locations specified by EA from the contents of the DAC. (See Chapter 6 of the System Architecture Reference Guide.) Loads the result in the DAC. An overflow causes a floating-point exception. If no floating-point exception occurs, CBIT is reset to 0. The values of LINK and the condition codes are indeterminate.

For 750 and 850 processors, exponent underflow is detected, but exponent overflow is not.

If a floating-point exception occurs and bit 7 of the keys contains a 1, the instruction sets CBIT to 1. If bit 7 contains a 0, the instruction sets CBIT to 1 and causes a floating-point exception fault. See Chapter 10 of the System Architecture Reference Guide for more information.

► DFST address  
 Double Precision Floating Point Store  
 I X 0 1 0 0 1 1 0 0 0 Y 1 0 BR\2 (V mode long)  
 DISPLACEMENT\16  
  
 I X 0 1 0 0 1 1 0 0 0 0 1 0 CB\2 (R mode long)  
 [ DISPLACEMENT\16 ]

Calculates an effective address, EA. Stores the contents of the DAC into the location specified by EA. (See Chapter 6 of the System Architecture Reference Guide.) Leaves the values of CBIT, LINK, and the condition codes unchanged.

#### Note

This instruction does not normalize the result before loading it into the specified memory location.

## ► DIV address

Divide

I X 1 1 1 1 1 1 0 0 0 0 0 0 CB\2 (R mode long)  
[ DISPLACEMENT\16 ]

I X 1 1 1 1 DISPLACEMENT\10 (S mode; R mode short)

Calculates an effective address, EA. Divides the 31-bit contents of A and B by the 16-bit contents of the location specified by EA. Stores the 16-bit quotient in A and the 16-bit remainder in B. The sign of the remainder equals the sign of the dividend.

Overflow occurs when the quotient is less than  $-(2^{15})$  or greater than  $(2^{15})-1$ . An overflow or a divide by 0 causes an integer exception. If no integer exception occurs, CBIT is reset to 0. This instruction leaves the values of LINK and the condition codes indeterminate.

If an integer exception occurs when bit 8 of the keys contains a 0, the instruction sets CBIT to 1. If bit 8 contains a 1, the instruction sets CBIT to 1 and causes an integer exception fault. See Chapter 10 of the System Architecture Reference Guide for more information.

## ► DIV address

Divide

I X 1 1 1 1 1 1 0 0 0 Y 0 0 BR\2 (V mode long)  
DISPLACEMENT\16

I X 1 1 1 1 DISPLACEMENT\10 (V mode short)

Calculates an effective address, EA. Divides the contents of L by the 16-bit contents of the location specified by EA. Stores the 16-bit quotient in A and the 16-bit remainder in B. The sign of the remainder equals the sign of the dividend.

When the quotient is less than  $-(2^{15})$  or greater than  $(2^{15})-1$ , an overflow occurs, causing an integer exception. A divide by 0 also causes an integer exception. If no integer exception occurs, CBIT is reset to 0. This instruction leaves the values of LINK and the condition codes indeterminate.

If the integer exception occurs when bit 8 of the keys is 0, the instruction sets CBIT to 1. If bit 8 is 1, the instruction sets CBIT to 1 and causes an integer exception fault. See Chapter 10 of the System Architecture Reference Guide for more information.

► DLD address  
 Double Load  
 I X 0 0 1 0 1 1 0 0 0 0 0 0 CB\2 (R mode long)  
 [ DISPLACEMENT\16 ]

I X 0 0 1 0 DISPLACEMENT\10 (S mode; R mode short)

Calculates an effective address, EA. Loads the 16-bit contents of the location specified by EA into A, and the 16-bit contents of the location specified by EA+1 into B. Leaves the values of CBIT, LINK, and the condition codes unchanged.

#### Note

This instruction executes only in double precision mode.

► DRN  
 Double Round From Quad  
 0 1 0 0 0 0 0 0 1 1 0 0 0 0 0 0 (V mode form)

Converts the value in QAC to a double precision floating-point number. If QAC contains 0, the instruction ends. If bits 50 to 96 of QAC are not 0, or bit 48 of QAC contains 1, the instruction adds the value of bit 49 to that of bit 48 (unbiased round) and clears bits 49 to 96 of QAC to 0. If any other condition exists, no unbiased rounding occurs but bits 49 to 96 of QAC are still cleared to 0. After any rounding and clearing occurs, the instruction normalizes the result and loads it into bits 1 to 64 of QAC.

If no floating-point exception occurs, the instruction resets CBIT to 0. The values of LINK and the condition codes are indeterminate.

If a floating-point exception occurs and bit 7 of the keys contains a 1, the instruction sets CBIT to 1. If bit 7 contains a 0, the instruction sets CBIT to 1 and causes a floating-point exception fault. See Chapter 10 of the System Architecture Reference Guide for more information.

#### Note

If DRN is used for any earlier system listed in "About This Book", an unimplemented instruction (UII) fault occurs. (See Chapter 10 of the System Architecture Reference Guide.)

## ► DRNM

Double Round From Quad Towards Negative Infinity  
 1 1 0 0 0 0 0 1 0 1 1 1 1 0 0 1 (V mode form)

Converts the value in QAC to a double precision floating-point number. If QAC contains 0, the instruction ends. If bits 49 to 96 of QAC contain zeros, the instruction ends. In any other case, the instruction clears bits 49 to 96 to 0, normalizes the result, and places it in bits 1 to 64 of QAC.

The value of CBIT is unchanged. The values of LINK and the condition codes are indeterminate.

Note

If DRNM is used for any earlier system listed in "About This Book", an unimplemented instruction (UII) fault occurs. (See Chapter 10 of the System Architecture Reference Guide.)

## ► DRNP

Double Round From Quad Towards Positive Infinity  
 0 1 0 0 0 0 0 0 1 1 0 0 0 0 0 1 (V mode form)

Converts the value in QAC to a double precision floating-point number. If QAC contains 0, the instruction ends. If bits 49 to 96 of QAC contain zeros, the instruction ends. In any other case, the instruction adds 1 to the value contained in bit 48 of QAC, clears bits 49 to 96 to 0, the instruction normalizes the result and places it in bits 1 to 64 of QAC.

If no floating-point exception occurs, the instruction resets CBIT to 0. The values of LINK and the condition codes are indeterminate.

If a floating-point exception occurs and bit 7 of the keys contains a 1, the instruction sets CBIT to 1. If bit 7 contains a 0, the instruction sets CBIT to 1 and causes a floating-point exception fault. See Chapter 10 of the System Architecture Reference Guide for more information.

Note

If DRNP is used for any earlier system listed in "About This Book", an unimplemented instruction (UII) fault occurs. (See Chapter 10 of the System Architecture Reference Guide.)

**DRNZ**

Double Round From Quad Towards Zero

0 1 0 0 0 0 0 0 1 1 0 0 0 0 1 0 (V mode form)

Converts the value in QAC to a double precision floating-point number. If QAC contains 0, the instruction ends. If bits 49 to 96 of QAC contain zeros and bit 1 contains 1, the instruction adds 1 to the value contained in bit 48 of QAC, clears bits 49 to 96 to 0, normalizes the result and places it in bits 1 to 64 of QAC. If any other condition exists, no rounding occurs.

If no floating-point exception occurs, the instruction resets CBIT to 0. The values of LINK and the condition codes are indeterminate.

If a floating-point exception occurs and bit 7 of the keys contains a 1, the instruction sets CBIT to 1. If bit 7 contains a 0, the instruction sets CBIT to 1 and causes a floating-point exception fault. See Chapter 10 of the System Architecture Reference Guide for more information.

Note

If DRNZ is used for any earlier system listed in "About This Book", an unimplemented instruction (UII) fault occurs. (See Chapter 10 of the System Architecture Reference Guide.)

**DRX**

Decrement and Replace X

1 1 0 0 0 0 0 0 1 0 0 0 1 0 0 0 (S, R, V mode form)

Decrements the contents of X by 1 and stores the result in X. Skips the next memory location if the decremented value is 0. Leaves the values of CBIT, LINK, and the condition codes unchanged.

**DSB address**

Double Subtract

 I X 0 1 1 1 1 1 0 0 0 0 0 0 0 CB\2 (R mode long)  
 [ DISPLACEMENT\16 ]

I X 0 1 1 1 DISPLACEMENT\10 (S mode; R mode short)

Calculates an effective address, EA. Fetches the 31-bit integer contained in the locations specified by EA and EA+1 and subtracts it from the 31-bit integer contained in A and B. Stores the result in A and B.

If the result is greater than or equal to  $2^{30}$ , an integer exception occurs and the DSB instruction loads bit 1 of A with 1, and bits 2 to 16 of A and 2 to 16 of B with the absolute value of  $(\text{result} - (2^{30}))$ . Bit 1 of B must be 0.

If the result is less than  $-(2^{30})$ , an integer exception occurs and the instruction loads bit 1 of A with a 0, and bits 2 to 16 of A and bits 2 to 16 of B with the negative of  $(\text{result} + (2^{30}))$ . Bit 1 of B must be 0.

If no integer exception occurs, CBIT is reset to 0. At the end of the instruction, LINK contains the borrow bit. The condition codes reflect the result of the operation. (See Appendix A.)

If an integer exception occurs and bit 8 of the keys contains 0, the instruction sets CBIT to 1. If bit 8 contains a 1, the instruction sets CBIT to 1 and causes an integer exception fault. See Chapter 10 of the System Architecture Reference Guide for more information.

#### Notes

1. Bit 17 of each 31-bit integer must be 0 or indeterminate results occur.
2. This instruction executes in double precision mode only.
3. To negate a 31-bit integer, subtract it from 0.

► DST address  
Double Store  
I X 0 1 0 0 1 1 0 0 0 0 0 0 CB\2 (R mode long)  
[ DISPLACEMENT\16 ]

I X 0 1 0 0 DISPLACEMENT\10 (S mode; R mode short)

Calculates an effective address, EA. Stores the contents of A at the location specified by EA, and the contents of B at the location specified by EA+1. Leaves the values of CBIT, LINK, and the condition codes unchanged.

#### Note

This instruction executes only in double precision mode.

► DVL address  
 Divide Long  
 I X 1 1 1 1 1 1 0 0 0 Y 1 1 BR\2 (V mode long)  
 DISPLACEMENT\16

Calculates an effective address, EA. Divides the 64-bit contents of L and E by the 32-bit contents of the location specified by EA. Stores the quotient in L and the remainder in E. An overflow or divide by 0 causes an integer exception. If no integer exception occurs, CBIT is reset to 0. The values of LINK and the condition codes are indeterminate.

If an integer exception occurs and bit 8 of the keys contains 0, the instruction sets CBIT to 1. If bit 8 contains a 1, the instruction sets CBIT to 1 and causes an integer exception fault. See Chapter 10 of the System Architecture Reference Guide for more information.

#### Note

This note applies only to the 150/250, 450/550/250-II, I450-II, and 2250 processors. When the value '040000 '000000 '000000 '000000 is divided by '100000 '000000, the quotient overflows the hardware (and sets the CBIT to 1) in the early stage of the algorithm even though the final result is not in overflow ('100000 '000000).

► E16S  
 Enter 16S Mode  
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 (S, R, V mode form)

Sets bits 4 to 6 of the keys to 000. Subsequent S mode instructions may now be interpreted, and 16S address calculations may be used to form effective addresses. Leaves the values of CBIT, LINK, and the condition codes unchanged.

► E32I  
 Enter 32I Mode  
 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 (S, R, V mode form)

Sets bits 4 to 6 of the keys to 100. Subsequent I mode instructions may now be interpreted, and 32I address calculations may be used to form effective addresses. Leaves the values of CBIT, LINK, and the condition codes unchanged.

► E32R  
 Enter 32R Mode  
 0 0 0 0 0 0 1 0 0 0 0 0 1 0 1 1 (S, R, V mode form)

Sets bits 4 to 6 of the keys to 011. Subsequent R mode instructions may now be interpreted, and 32R address calculations may be used to form effective addresses. Leaves the values of CBIT, LINK, and the condition codes unchanged.

► E32S  
 Enter 32S Mode  
 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 1 (S, R, V mode form)

Sets bits 4 to 6 of the keys to 001. Subsequent S mode instructions may now be interpreted, and 32S address calculations may be used to form effective addresses. Leaves the values of CBIT, LINK, and the condition codes unchanged.

► E64R  
 Enter 64R Mode  
 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 1 (S, R, V mode form)

Sets bits 4 to 6 of the keys to 010. Subsequent R mode instructions may now be interpreted, and 64R address calculations may be used to form effective addresses. Leaves the values of CBIT, LINK, and the condition codes unchanged.

► E64V  
 Enter 64V Mode  
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 (S, R, V mode form)

Sets bits 4 to 6 of the keys to 110. Subsequent V mode instructions may now be interpreted, and 64V address calculations may be used to form effective addresses. Leaves the values of CBIT, LINK, and the condition codes unchanged.

► EAA address  
 Effective Address to A  
 I X 0 0 0 1 1 1 0 0 0 0 0 1 CB\2 (R mode form)  
 DISPLACEMENT\16

Calculates an effective address, EA, and loads it into A. Leaves the values of CBIT, LINK, and the condition codes unchanged.

► EAFA far,address  
 Effective Address to FAR  
 0 0 0 0 0 0 1 0 1 1 0 0 FAR 0 0 0 (V mode form)  
 AP\32

Builds a 36-bit EA from the 32-bit address pointer contained in the instruction and loads it into the specified FAR. The AP bit field is processed and loaded into the bit portion of the FAR for direct access; indirection uses the bit field in the indirect pointer (if any). Leaves the values of CBIT, LINK, and the condition codes unchanged.

Figure 2-3 shows the format of the EA loaded into the specified FAR.

1	16 17	32 33	36
RING, SEG	WORD #	BIT #	

EA Format for EAFA  
 Figure 2-3

► EAL address  
 Effective Address to L  
 I X 0 0 0 1 1 1 0 0 0 Y 0 1 BR\2 (V mode form)  
 DISPLACEMENT\16

Calculates an effective address, EA, and loads it into L. Leaves the values of CBIT, LINK, and the condition codes unchanged.

► **EALB address**  
 Effective Address to LB  
 I X 1 0 1 1 1 1 0 0 0 Y 1 0 BR\2 (V mode form)  
 DISPLACEMENT\16

Calculates an effective address, EA, and loads it into LB. Leaves the values of CBIT, LINK, and the condition codes unchanged.

► **EAXB address**  
 Effective Address to XB  
 I X 1 0 1 0 1 1 0 0 0 Y 1 0 BR\2 (V mode form)  
 DISPLACEMENT\16

Calculates an effective address, EA, and loads it into XB. Leaves the values of CBIT, LINK, and the condition codes unchanged.

► **EIO address**  
 Execute I/O  
 I 0 1 1 0 0 1 1 0 0 0 Y 0 1 BR\2 (V mode form)  
 DISPLACEMENT\16

Calculates an effective address, EA. Executes bits 17 to 32 of EA as if the bits were an extended PIO instruction. If execution is successful, the instruction sets the condition codes as follows:

<u>CC</u>	<u>Meaning</u>
EQ	Successful INA, OTA, or SKS instruction
NE	Unsuccessful INA, OTA, or SKS; all OCP

Leaves the values of LINK and CBIT unchanged. See Chapter 11 of the System Architecture Reference Guide for more information.

#### Note

This is a restricted instruction.

► **ENB**  
 Enable Interrupts  
 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 (S, R, V mode form)

Enables interrupts by setting bit 1 of the modals to 1. Interrupts remain inhibited for the next instruction. Leaves the values of CBIT, LINK, and the condition codes unchanged.

Note

ENB is a restricted instruction.

► ENBL  
 Enable Interrupts (Local)  
 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 (S, R, V mode form)

This 850 instruction performs the same actions as ENB except that it is performed specifically for the local processor. Leaves the values of CBIT, LINK, and the condition codes unchanged.

Note

ENBL is a restricted instruction.

► ENBM  
 Enable Interrupts (Mutual)  
 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 (S, R, V mode form)

For the 850, a processor checks the availability of the mutual exclusion lock. If available, the processor releases this lock and enables interrupts. Leaves the values of CBIT, LINK, and the condition codes unchanged.

Note

This is a restricted instruction.

► ENBP  
 Enable Interrupts (Process)  
 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 (S, R, V mode form)

For the 850, a processor checks the availability of the process exchange lock. If available, the processor releases this lock and enables interrupts. Leaves the values of CBIT, LINK, and the condition codes unchanged.

Note

This is a restricted instruction.

► ERA address  
 Exclusive OR to A  
 I X 0 1 0 1 1 1 0 0 0 Y 0 0 BR\2 (V mode long)  
 DISPLACEMENT\16

I X 0 1 0 1 1 1 0 0 0 0 0 0 0 CB\2 (R mode long)  
 [ DISPLACEMENT\16 ]

I X 0 1 0 1 DISPLACEMENT\10 (S mode; R, V mode short)

Calculates an effective address, EA. Exclusively ORs the contents of the location specified by EA and the contents of A. Stores the results in A. Leaves the values of CBIT, LINK, and the condition codes unchanged.

► ERL address  
 Exclusive Or to L  
 I X 0 1 0 1 1 1 0 0 0 Y 1 1 BR\2 (V mode long)  
 DISPLACEMENT\16

Calculates an effective address, EA. Exclusively ORs the contents of L with the contents of the 32-bit location specified by EA. Stores the results in L. Leaves the values of CBIT, LINK, and the condition codes unchanged.

► FAD address  
 Floating Point Add  
 I X 0 1 1 0 1 1 0 0 0 Y 0 1 ER\2 (V mode long)  
 DISPLACEMENT\16  
  
 I X 0 1 1 0 1 1 0 0 0 0 1 CB\2 (R mode long)  
 [ DISPLACEMENT\16 ]

Calculates an effective address, EA. Adds the contents of the location specified by EA to the contents of the FAC. (See Chapter 6 of the System Architecture Reference Guide.) Stores the result in the FAC and normalizes it if necessary. An overflow causes a floating-point exception. If no floating-point exception occurs, CBIT is reset to 0. The values of LINK and the condition codes are indeterminate.

If a floating-point exception occurs and bit 7 of the keys contains a 1, the instruction sets CBIT to 1. If bit 7 contains a 0, the instruction sets CBIT to 1 and causes a floating-point exception fault. See Chapter 10 of the System Architecture Reference Guide for more information.

► FCDQ  
 Floating Point Convert Double to Quad  
 1 1 0 0 0 0 0 1 0 1 1 1 1 0 0 1 (V mode form)

Clears FAC1 to 0. Leaves the values of CBIT, LINK, and the condition codes unchanged.

#### Note

If FCDQ is used for any earlier system listed in "About This Book", an unimplemented instruction (UII) fault occurs. (See Chapter 10 of the System Architecture Reference Guide.)

► FCM  
 Floating Point Complement  
 1 1 0 0 0 0 0 1 0 1 0 1 1 0 0 0 (R, V mode form)

Forms the two's complement of the FAC mantissa and normalizes the result if necessary. (See Chapter 6 of the System Architecture Reference Guide.) Stores the result in the FAC. An overflow causes a floating-point exception. If no floating-point exception occurs, CBIT is reset to 0. The values of LINK and the condition codes are indeterminate.

If a floating-point exception occurs and bit 7 of the keys contains a 1, the instruction sets CBIT to 1. If bit 7 contains a 0, the instruction sets CBIT to 1 and causes a floating-point exception fault. See Chapter 10 of the System Architecture Reference Guide.

► FCS address  
 Floating Point Compare and Skip  
 I X 1 0 0 1 1 1 0 0 0 Y 0 1 BR\2 (V mode long)  
 DISPLACEMENT\16

I X 1 0 0 1 1 1 0 0 0 0 0 1 CB\2 (R mode long)  
 [ DISPLACEMENT\16 ]

Calculates an effective address, EA. In rounding mode, the instruction rounds the contents of DAC, then compares the rounded value to the contents of the memory location specified by EA. In normal mode, no rounding occurs before the compare. (See Chapter 6 of the System Architecture Reference Guide for more information.) The compare results in a skip as follows:

<u>Condition</u>	<u>Skip</u>
FAC contents > EA contents.	No skip.
FAC contents = EA contents.	Skip 16 bits (one halfword).
FAC contents < EA contents.	Skip 32 bits (two halfwords).

The values of CBIT, LINK, and the condition codes are indeterminate.

On some processors, FCS works correctly only on normalized numbers as follows. The comparison has a maximum of three sequential stages: first the signs, then the exponents, and finally the fractions of the two numbers are compared for equality. If the comparison during any one of these stages reveals an inequality, the results are returned and the instruction ends. Unnormalized numbers are unexpected and produce unexpected results. Other processors actually perform a subtract, resulting in a proper comparison.

► FDBL  
 Floating Point Convert Single to Double  
 1 1 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 (V mode form)

Converts the single precision floating-point number in the floating accumulator to a double precision floating-point number by loading zeros into bits 33 to 48 of the floating accumulator. Leaves the values of CBIT, LINK, and the condition codes unchanged.

► **FDV address**  
 Floating Point Divide  
 I X 1 1 1 1 1 1 0 0 0 Y 0 1 BR\2 (V mode long)  
 DISPLACEMENT\16

I X 1 1 1 1 1 1 0 0 0 0 0 1 CB\2 (R mode long)  
 [ DISPLACEMENT\16 ]

Calculates an effective address, EA. Divides the contents of the FAC by the contents of the location specified by EA. (See Chapter 6 of the System Architecture Reference Guide.) Normalizes the result if necessary and stores it in the FAC. A divide by 0 or an overflow causes a floating-point exception. If no floating-point exception occurs, CBIT is reset to 0. The values of LINK and the condition codes are indeterminate.

If a floating-point exception occurs and bit 7 of the keys contains a 1, the instruction sets CBIT to 1. If bit 7 contains a 0, the instruction sets CBIT to 1 and causes a floating-point exception fault. See Chapter 10 of the System Architecture Reference Guide for more information.

#### Note

The location specified by EA must contain a normalized floating-point number. An unnormalized divisor can cause an error.

► **FLD address**  
 Floating Point Load  
 I X 0 0 1 0 1 1 0 0 0 Y 0 1 BR\2 (V mode long)  
 DISPLACEMENT\16

I X 0 0 1 0 1 1 0 0 0 0 0 1 CB\2 (R mode long)  
 [ DISPLACEMENT\16 ]

Calculates a 32-bit effective address, EA. Loads the 32-bit contents in the location specified by EA into the FAC without normalizing. (See Chapter 6 of the System Architecture Reference Guide.) Leaves the values of LINK, CBIT, and the condition codes unchanged.

► **FLOT**  
 Convert Integer to Floating Point  
 1 1 0 0 0 0 0 1 0 1 1 0 1 0 0 0 (R mode form)

Converts the 31-bit integer contained in A and B to a normalized floating-point number and stores the result in the floating accumulator. The values of CBIT, LINK, and the condition codes are indeterminate.

## ► FLTA

Convert Integer to Floating Point

1 1 0 0 0 0 0 1 0 1 0 1 1 0 1 0 (V mode form)

Converts the 16-bit integer in A to a floating-point number and stores the result in the floating accumulator. The values of CBIT, LINK, and the condition codes are indeterminate.

## ► FLTL

Convert Long Integer to Floating Point

1 1 0 0 0 0 0 1 0 1 0 1 1 1 0 1 (V mode form)

Converts the 32-bit integer in L to a floating-point number and stores the result in the floating accumulator. The values of CBIT, LINK, and the condition codes are indeterminate.

## ► FLX address

Floating Load Index

I 0 1 1 0 1 1 1 0 0 0 Y 0 1 BR\2 (V mode long)  
DISPLACEMENT\16I 0 1 1 0 1 1 1 0 0 0 0 1 CB\2 (R mode long)  
[ DISPLACEMENT\16 ]

Calculates an effective address, EA. Loads the index register, X, with two times the 16-bit contents of the location specified by EA. Leaves the values of CBIT, LINK, and the condition codes unchanged.

Note

FLX cannot do indexing. See Appendix B for more information.

## ► FMP address

Floating Point Multiply

I X 1 1 1 0 1 1 0 0 0 Y 0 1 BR\2 (V mode long)  
DISPLACEMENT\16I X 1 1 1 0 1 1 0 0 0 0 1 CB\2 (R mode long)  
[ DISPLACEMENT\16 ]

Calculates an effective address, EA. Multiplies the contents of the FAC by the contents of the location specified by EA. (See Chapter 6 of the System Architecture Reference Guide.) Normalizes the result if necessary and stores it in the FAC. An overflow causes a floating-point exception. If no floating-point exception occurs, CBIT is reset to 0. The values of LINK and the condition codes are indeterminate.

## INSTRUCTION SETS GUIDE

If a floating-point exception occurs and bit 7 of the keys contains a 1, the FMP instruction sets CBIT to 1. If bit 7 contains a 0, the instruction sets CBIT to 1 and causes a floating-point exception fault. See Chapter 10 of the System Architecture Reference Guide for more information.

► **FRN**  
Floating Point Round  
1 1 0 0 0 0 0 1 0 1 0 1 1 1 0 0 (R, V mode form)

This instruction operates on and stores all results in the floating accumulator.

For the 2350 to the 9955 II, the following actions occur. If bits 1 to 48 contain 0, then bits 49 to 64 are cleared to 0. If bits 24 and 25 both contain 1, then 1 is added to bit 24, bits 25 to 48 are cleared to 0, and the result is normalized. If bit 25 contains 1 and bits 26 to 48 are not equal to 0, then 1 is added to bit 24, bits 25 to 48 are cleared, and the result is normalized.

For the earlier systems listed in "About This Book", the following actions occur. If bits 1 to 48 contain 0, then bits 49 to 64 are cleared to 0. Otherwise, bit 25 is added to bit 24, bits 25 to 48 are cleared to 0, and the result is normalized.

For all systems, if no floating point exception occurs, sets CBIT to 0. The values of LINK and the condition codes are indeterminate.

If a floating point exception occurs and bit 7 of the keys contains a 1, the instruction sets CBIT to 1. If bit 7 contains a 0, the instruction sets CBIT to 1 and causes a floating point exception fault. See Chapter 10 of the System Architecture Reference Guide for more information.

► **FRNM**  
Floating Point Round Towards Negative Infinity  
0 1 0 0 0 0 0 0 1 1 0 1 0 0 0 0 (V mode form)

Converts the 64-bit value in DAC to a single precision floating-point number. If DAC contains 0, the instruction ends. If bits 25 to 48 of DAC contain zeros, the instruction ends. In any other case, the instruction clears bits 25 to 48 to 0, normalizes the result, and places it in DAC. If no floating-point exception occurs, the instruction resets CBIT to 0. The values of LINK and the condition codes are indeterminate.

If a floating-point exception occurs and bit 7 of the keys contains a 1, the instruction sets CBIT to 1. If bit 7 contains a 0, the instruction sets CBIT to 1 and causes a floating-point exception fault. (See Chapter 10 of the System Architecture Reference Guide.)

## ► FRNP

Floating Point Round Towards Positive Infinity  
 0 1 0 0 0 0 0 0 1 1 0 0 0 0 1 1 (V mode form)

Converts the 64-bit value in DAC to a single precision floating-point number. If DAC contains 0, the instruction ends. If bits 25 to 48 of DAC contain zeros, the instruction ends. In any other case, the instruction adds 1 to the value contained in bit 24 of DAC, clears bits 25 to 48 to 0, normalizes the result, and places it in DAC.

If no floating-point exception occurs, the instruction resets CBIT to 0. The values of LINK and the condition codes are indeterminate.

If a floating-point exception occurs and bit 7 of the keys contains a 1, the instruction sets CBIT to 1. If bit 7 contains a 0, the instruction sets CBIT to 1 and causes a floating-point exception fault. See Chapter 10 of the System Architecture Reference Guide for more information.

## ► FRNZ

Floating Point Round Towards Zero  
 0 1 0 0 0 0 0 0 1 1 0 1 0 0 0 1 (V mode form)

Converts the 64-bit value in DAC to a single precision floating-point number. If DAC contains 0, the instruction ends. If bits 25 to 48 of DAC are not zeros and bit 1 contains 1, the instruction adds 1 to the value contained in bit 24 of DAC, clears bits 25 to 48 to zero, normalizes the result, and places it in DAC. If any other condition exists, no rounding occurs.

If no floating-point exception occurs, the instruction resets CBIT to 0. The values of LINK and the condition codes are indeterminate.

If a floating-point exception occurs and bit 7 of the keys contains a 1, the instruction sets CBIT to 1. If bit 7 contains a 0, the instruction sets CBIT to 1 and causes a floating-point exception fault. See Chapter 10 of the System Architecture Reference Guide for more information.

► FSB address  
 Floating Point Subtract  
 I X 0 1 1 1 1 0 0 0 Y 0 1 BR\2 (V mode long)  
 DISPLACEMENT\16  
  
 I X 0 1 1 1 1 0 0 0 0 0 1 CB\2 (R mode long)  
 [ DISPLACEMENT\16 ]

Calculates an effective address, EA. Subtracts the 32-bit contents of the locations specified by EA from the contents of the FAC. (See Chapter 6 of the System Architecture Reference Guide.) Normalizes the result if necessary and stores it in the FAC. An overflow causes a floating-point exception. If no floating-point exception occurs, CBIT is reset to 0. The values of LINK and the condition codes are indeterminate.

If a floating-point exception occurs and bit 7 of the keys contains a 1, the FSB instruction sets CBIT to 1. If bit 7 contains a 0, the instruction sets CBIT to 1 and causes a floating-point exception fault. See Chapter 10 of the System Architecture Reference Guide for more information.

► FSGT  
 Floating Point Skip on F Greater Than 0  
 1 1 0 0 0 0 0 1 0 1 0 0 1 1 0 1 (R, V mode form)

Skips the next 16-bit halfword if the contents of the floating accumulator are greater than 0. Leaves the value of LINK and CBIT unchanged. The values of the condition codes are indeterminate. FSGT works correctly only on normalized or nearly normalized numbers, because it checks the first 32 fraction bits only for equal to zero and less than zero. (See Chapter 6 in the System Architecture Reference Guide.)

► FSLE  
 Floating Point Skip on F Less Than or Equal to 0  
 1 1 0 0 0 0 0 1 0 1 0 0 1 1 0 0 (R, V mode form)

Skips the next 16-bit halfword if the contents of the floating accumulator are less than or equal to 0. Leaves the values of LINK and CBIT unchanged. The values of the condition codes are indeterminate. FSLE works correctly only on normalized or nearly normalized numbers, because it checks the first 32 fraction bits only for equal to zero and less than zero. (See Chapter 6 in the System Architecture Reference Guide.)

## ► FSMI

Floating Point Skip on F Minus

1 1 0 0 0 0 0 1 0 1 0 0 1 0 1 0 (R, V mode form)

Skips the next 16-bit halfword if the contents of the floating accumulator are less than 0. Leaves the values of LINK and CBIT unchanged. The values of the condition codes are indeterminate. FSMI works correctly only on normalized or nearly normalized numbers, because it checks the first 32 fraction bits only for equal to zero and less than zero. (See Chapter 6 in the System Architecture Reference Guide.)

## ► FSNZ

Floating Point Skip on F Not 0

1 1 0 0 0 0 0 1 0 1 0 0 1 0 0 1 (R, V mode form)

Skips the next 16-bit halfword if the contents of the floating accumulator are less than or equal to 0. Leaves the values of LINK and CBIT unchanged. The values of the condition codes are indeterminate. FSNZ works correctly only on normalized or nearly normalized numbers, because it checks the first 32 fraction bits only for equal to zero and less than zero. (See Chapter 6 in the System Architecture Reference Guide.)

## ► FSPL

Floating Point Skip on FAC Plus

1 1 0 0 0 0 0 1 0 1 0 0 1 0 1 1 (R, V mode form)

Skips the next 16-bit halfword if the contents of the floating accumulator are greater than or equal to 0. Leaves the values of LINK and CBIT unchanged. The values of the condition codes are indeterminate. FSPL works correctly only on normalized or nearly normalized numbers, because it checks the first 32 fraction bits only for equal to zero and less than zero. (See Chapter 6 in the System Architecture Reference Guide.)

## ► FST address

Floating Point Store

I X 0 1 0 0 1 1 0 0 0 Y 0 1 BR\2 (V mode long)  
DISPLACEMENT\16I X 0 1 0 0 1 1 0 0 0 0 0 1 CB\2 (R mode long)  
[ DISPLACEMENT\16 ]

Calculates an effective address, EA. Stores the contents of the FAC into the 32-bit location specified by EA. (See Chapter 6 of the System Architecture Reference Guide.) If the exponent contained in the FAC is too large to be expressed in 8 bits, a floating-point exception (store

exception) occurs. If no floating-point exception occurs, the instruction resets CBIT to 0. At the end of the instruction, the values of LINK and the condition codes are indeterminate.

If a floating-point exception occurs and bit 7 of the keys contains a 1, the instruction sets CBIT to 1. If bit 7 contains a 0, the instruction sets CBIT to 1 and causes a floating-point exception fault. See Chapter 10 of the System Architecture Reference Guide for more information. In either case, a floating-point exception leaves the contents of the memory location in an indeterminate state.

This instruction does not normalize the result before loading it into the specified memory location unless rounding is enabled.



#### FSZE

Floating Point Skip on F Equal to 0

1 1 0 0 0 0 0 1 0 1 0 0 1 0 0 0 (R, V mode form)

Skips the next 16-bit halfword if the contents of the floating accumulator equal 0. Leaves the values of LINK and CBIT unchanged. The values of the condition codes are indeterminate. FSZE works correctly only on normalized or nearly normalized numbers, because it checks the first 32 fraction bits only for equal to zero and less than zero. (See Chapter 6 in the System Architecture Reference Guide.)

► **HLT**  
 Halt  
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 (S, R, V mode form)

Halts computer operation. The program counter points to the instruction that would have been executed if execution had not been stopped. The supervisor terminal indicates a halt. Leaves the values of CBIT, LINK, and the condition codes unchanged.

This instruction saves the contents of registers in a memory location specified by the RSAVPTR. The contents of RSAVPTR can be accessed by an LDIR/STLR instruction with address '40037. The registers are saved in their physical order. (See Chapter 9 of the System Architecture Reference Guide for the format of these register files.) The saved register file order is shown in Table 2-3.

Table 2-3  
 Order of Saved Registers after HLT

6350, 9750 to 9955 II	2350 to 2755, 9650 and 9655	Earlier Systems*
User Reg Set 3	User Reg Set 1	User Reg Set 2
User Reg Set 4	User Reg Set 2	User Reg Set 1
User Reg Set 1	User Reg Set 3	DMx Reg File
User Reg Set 2	User Reg Set 4	Microcode Reg File
Microcode Reg File, Set 2	User Reg Set 5	
Indirect Reg Set	User Reg Set 6	
Microcode Reg File, Set 1	User Reg Set 7	
DMx Reg File	User Reg Set 8	
	DMx Reg File	
	Microcode Reg File, Set 1	
	Microcode Reg File, Set 2	

\* The earlier systems are listed in "About This Book". Of these, the 850 has two ISPs. For each ISP, the order of saved registers is identical to the order shown for the rest of the 50 Series.

#### Note

This is a restricted instruction.

► IAB  
Interchange A and B  
0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 (S, R, V mode form)

Interchanges the contents of A and B. Leaves the values of LINK, CBIT, and the condition codes unchanged.

► ICA  
Interchange Bytes of A Register  
1 1 0 0 0 0 1 0 1 1 1 0 0 0 0 0 (S, R, V mode form)

Interchanges the bytes of A. Leaves the values of CBIT, LINK, and the condition codes unchanged.

► ICL  
Interchange Bytes and Clear Left  
1 1 0 0 0 0 1 0 0 1 1 0 0 0 0 0 (S, R, V mode form)

Interchanges the bytes of A, then clears the left byte to 0. Leaves the values of CBIT, LINK, and the condition codes unchanged.

► ICR  
Interchange Bytes and Clear Right  
1 1 0 0 0 0 1 0 1 0 1 0 0 0 0 0 (S, R, V mode form)

Interchanges the bytes of A, then clears the right byte to 0. Leaves the values of CBIT, LINK, and the condition codes unchanged.

► ILE  
Interchange L and E  
1 1 0 0 0 0 1 1 0 0 0 0 1 1 0 0 (S, R, V mode form)

Interchanges the values of E and L. Leaves the values of CBIT, LINK, and the condition codes unchanged.

- IMA address  
Interchange Memory and A  
I X 1 0 1 1 1 1 0 0 0 Y 0 0 BR\2 (V mode long)  
DISPLACEMENT\16  
  
I X 1 0 1 1 1 1 0 0 0 0 0 0 CB\2 (R mode long)  
[ DISPLACEMENT\16 ]  
  
I X 1 0 1 1 DISPLACEMENT\10 (S mode; R, V mode short)

Calculates an effective address, EA. Interchanges the contents of A and the contents of the location specified by EA. Leaves the values of CBIT, LINK, and the condition codes unchanged.

#### Note

The IMA instruction is nonatomic, and, especially for dual-stream processors, cannot be used for spin-locks. In these cases, use the STAC instruction instead.

- INA function,device  
Input to A  
1 0 1 1 0 0 FUNCTION\4 DEVICE\6  
Valid for modes S, R

Loads data from the specified device into A. Leaves the values of CBIT, LINK, and the condition codes unchanged. See Chapter 11 of the System Architecture Reference Guide for more information.

#### Note

This is a restricted instruction.

- INBC address  
Interrupt Notify Beginning, Clear Active Interrupt  
0 0 0 0 0 1 0 1 0 0 0 1 1 1 1 (V mode form)  
AP\32

Notifies a semaphore at the specified address during phantom interrupt code. Restores the state of the interrupted process by loading bits 1 to 16 of PB, bits 17 to 32 of the program counter, and the keys from microcode temporary registers PSWPB and PSWKEYS. Places the notified process at the beginning of the appropriate priority level queue. Issues a CAI pulse to clear the currently active interrupt, and enables interrupts. The values of CBIT, LINK, and the condition codes are indeterminate. A process exchange will occur if the notified process is of a higher priority than the interrupted process. See Chapter 9 of the System Architecture Reference Guide for more information.

Note

INBC is a restricted instruction.

This instruction is normally used to transfer from phantom interrupt code to an interrupt process. See Chapter 10 of the System Architecture Reference Guide for more information.

► INEN address  
 Interrupt Notify Beginning  
 0 0 0 0 0 1 0 1 0 0 0 1 1 0 1 (V mode form)  
 AP\32

Notifies a semaphore at the specified address during phantom interrupt code. Restores the state of the interrupted process by loading bits 1 to 16 of PB, bits 17 to 32 of the program counter, and the keys from microcode temporary registers PSWPB and PSWKEYS. Places the notified process at the beginning of the appropriate priority level queue, and enables interrupts. Does not issue a CAI pulse to clear the currently active interrupt. The values of CBIT, LINK, and the condition codes are indeterminate. A process exchange will occur if the notified process is of a higher priority than the interrupted process. See Chapter 9 of the System Architecture Reference Guide for more information.

Note

This is a restricted instruction.

This instruction is normally used to transfer from phantom interrupt code to an interrupt process. See Chapter 10 of the System Architecture Reference Guide for more information.

► INEC address  
 Interrupt Notify End, Clear Active Interrupt  
 0 0 0 0 0 1 0 1 0 0 0 1 1 1 0 (V mode form)  
 AP\32

Notifies a semaphore at the specified address during phantom interrupt code. Restores the state of the interrupted process by loading bits 1 to 16 of PB, bits 17 to 32 of the program counter, and the keys from microcode temporary registers PSWPB and PSWKEYS. Places the notified process at the end of the appropriate priority level queue. Issues a CAI pulse to clear the currently active interrupt, and enables interrupts. The values of CBIT, LINK, and the condition codes are indeterminate. A process exchange will occur if the notified process is of a higher priority than the interrupted process. See Chapter 9 of the System Architecture Reference Guide for more information.

Note

INEC is a restricted instruction.

This instruction is normally used to transfer from phantom interrupt code to an interrupt process. See Chapter 10 of the System Architecture Reference Guide for more information.

► INEN address  
 Interrupt Notify End  
 0 0 0 0 0 0 1 0 1 0 0 0 1 1 0 0 (V mode form)  
 AP\32

Notifies a semaphore at the specified address during phantom interrupt code. Restores the state of the interrupted process by loading bits 1 to 16 of PB, bits 17 to 32 of the program counter, and the keys from microcode temporary registers PSWPB and PSWKEYS. Places the notified process at the end of the appropriate priority level queue, and enables interrupts. Does not issue a CAI pulse to clear the currently active interrupt. The values of CBIT, LINK, and the condition codes are indeterminate. A process exchange will occur if the notified process is of a higher priority than the interrupted process. See Chapter 9 of the System Architecture Reference Guide for more information.

Note

This is a restricted instruction.

This instruction is normally used to transfer from phantom interrupt code to an interrupt process. See Chapter 10 of the System Architecture Reference Guide for more information.

► INH  
 Inhibit Interrupts  
 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 (S, R, V mode form)

Inhibits interrupts by setting bit 1 of the modals to 0. Inhibits interrupts until an enable interrupts instruction executes. The processor ignores any interrupt requests that are made over the I/O bus. This instruction takes effect immediately. Leaves the values of CBIT, LINK, and the condition codes unchanged.

Note

This is a restricted instruction.

► **INHL**  
 Inhibit Interrupts (Local)  
 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 (S, R, V mode form)

This 850 instruction performs the same actions as INH does. Leaves the values of CBIT, LINK, and the condition codes unchanged.

Note

This is a restricted instruction.

► **INHM**  
 Inhibit Interrupts (Mutual)  
 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 (S, R, V mode form)

For the 850, a processor checks the availability of the mutual exclusion lock. If available, the processor sets this lock and inhibits interrupts. Otherwise, it waits for the lock to be released by the other processor and then sets the lock and inhibits interrupts. Leaves the values of CBIT, LINK, and the condition codes unchanged.

Note

This is a restricted instruction.

► **INHP**  
 Inhibit Interrupts (Process)  
 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 (S, R, V mode form)

For the 850, a processor checks the availability of the process exchange lock. If available, the processor sets it and inhibits interrupts. Otherwise, it waits for the lock to be released by the other processor and then sets the lock and inhibits interrupts. Leaves the values of CBIT, LINK, and the condition codes unchanged.

Note

This is a restricted instruction.

► **INK**  
 Input Keys  
 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 1 (S, R mode form)

Loads the contents of the S and R mode keys into A. Reads the low-order 8 bits of the floating exponent (address trap location 6) register along with the high-order 8 bits of the keys register. Leaves the values of CBIT, LINK, and the condition codes unchanged.

► **INT**  
 Convert Floating Point to Integer  
 1 1 0 0 0 0 0 1 0 1 1 0 1 1 0 0 (S, R mode form)

Converts the double precision floating-point number contained in the floating accumulator to a 31-bit integer and stores the result in A and bits 2 to 16 of B. Bit 1 of B (bit 17 of the result) is forced to 0. Ignores the fractional portion of the floating-point number. Overflow occurs if the value in the floating accumulator is less than  $-2^{30}$  or greater than  $(2^{30})-1$ . If overflow occurs, a floating-point exception occurs. If no floating-point exception occurs, CBIT is reset to 0. The values of LINK and the condition codes are indeterminate.

If a floating-point exception occurs and bit 7 of the keys contains a 1, the instruction sets CBIT to 1. If bit 7 contains a 0, the instruction sets CBIT to 1 and causes a floating-point exception fault. See Chapter 10 of the System Architecture Reference Guide for more information.

► **INTA**  
 Convert Floating Point to Integer  
 1 1 0 0 0 0 0 1 0 1 0 1 1 0 0 1 (V mode form)

Converts the double precision number contained in the floating accumulator to a 16-bit integer and stores the result in A. Ignores the fractional portion of the floating-point number. For example, -4.5 is converted to -4 and +4.5 is converted to +4. Overflow occurs if the value in the floating accumulator is less than  $-2^{15}$  or greater than  $(2^{15})-1$ . If overflow occurs, a floating-point exception occurs. If no floating-point exception occurs, CBIT is reset to 0.

At the end of this instruction, the B register contents are indeterminate. The values of LINK and the condition codes are indeterminate.

If a floating-point exception occurs and bit 7 of the keys contains a 1, the instruction sets CBIT to 1. If bit 7 contains a 0, the instruction sets CBIT to 1 and causes a floating-point exception fault. See Chapter 10 of the System Architecture Reference Guide for more information.

► INTL  
 Convert Floating Point to Long Integer  
 1 1 0 0 0 0 0 1 0 1 0 1 1 0 1 1 (V mode form)

Converts the double precision floating-point number contained in the floating accumulator to a 32-bit integer and stores the result in L. Ignores the fractional portion of the floating-point number contained in the floating accumulator. For example, -4.5 is converted to -4 and +4.5 is converted to +4. Overflow occurs if the floating-point number is less than  $-2^{*}31$  or greater than  $(2^{*}31)-1$ . If overflow occurs, a floating-point exception occurs. If no floating-point exception occurs, CBIT is reset to 0. The values of LINK and the condition codes are indeterminate.

If a floating-point exception occurs and bit 7 of the keys contains a 1, the instruction sets CBIT to 1. If bit 7 contains a 0, the instruction sets CBIT to 1 and causes a floating-point exception fault. See Chapter 10 of the System Architecture Reference Guide for more information.

► IRS address  
 Increment and Replace Memory  
 I X 1 0 1 0 1 1 0 0 0 Y 0 0 ER\2 (V mode long)  
 DISPLACEMENT\16  
  
 I X 1 0 1 0 1 1 0 0 0 0 0 0 CB\2 (R mode long)  
 [ DISPLACEMENT\16 ]  
  
 I X 1 0 1 0 DISPLACEMENT\10 (S mode; R, V mode short)

Calculates an effective address, EA. Fetches the contents of the location specified by EA, adds 1 (a 16-bit increment), and stores the result back in the location specified by EA. Skips the next location if the incremented value is 0. Leaves the values of CBIT, LINK, and the condition codes unchanged.

► IRTC  
 Interrupt Return, Clear Active Interrupt  
 0 0 0 0 0 0 0 1 1 0 0 0 0 0 1 1 (V mode form)

Returns from an interrupt. Restores the state existing before the interrupt by loading bits 1 to 16 of PB, bits 17 to 32 of the program counter, and the keys from the values saved in microcode temporary registers PSWPB and PSWKEYS. Issues a CAI pulse to clear the currently active interrupt, and enables interrupts.

#### Note

This is a restricted instruction.

► **IRTN**  
 Interrupt Return  
 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 1 (V mode form)

Returns from an interrupt. Restores the state existing before the interrupt by loading bits 1 to 16 of PB, bits 17 to 32 of the program counter, and the keys from the values saved in microcode temporary registers PSWPB and PSWKEYS, and enables interrupts. Does not issue a CAI pulse to clear the currently active interrupt.

#### Note

This is a restricted instruction.

► **IRX**  
 Increment and Replace X  
 1 1 0 0 0 0 0 0 0 1 0 0 1 1 0 0 (S, R, V mode form)

Increments the contents of X by 1 and stores the result in X. Skips the next 16-bit halfword if the incremented value is 0. Leaves the values of CBIT, LINK, and the condition codes unchanged.

► **ITLB**  
 Invalidate STLB Entry  
 0 0 0 0 0 0 0 1 1 0 0 0 1 1 0 1 (V mode form)

Invalidates the STLB entry that corresponds to the virtual address contained in L. The values of CBIT, LINK, and the condition codes are indeterminate. You must execute this instruction whenever you change the page table entry for the given address.

If you change an SDW or DTAR (explained in Chapter 4 of the System Architecture Reference Guide), you usually have to invalidate the entire STLB by issuing the instruction PTLB. A 0 in the segment number portion of L invalidates the IOTLB entry corresponding to the address specified by L.

#### Note

This is a restricted instruction.

► JDX address  
 Jump and Decrement X  
 I 0 1 1 0 1 1 1 0 0 0 0 1 0 CB\2 (R mode long)  
 [ DISPLACEMENT\16 ]

Calculates an effective address, EA. Subtracts 1 from the contents of the index register, X. If the decremented value does not equal 0, the instruction loads EA into the program counter. If the decremented value is equal to 0, execution continues with the next sequential instruction. Leaves the values of CBIT, LINK, and the condition codes unchanged.

#### Note

This instruction cannot do indexing. See Appendix B for more information.

► JIX address  
 Increment X and Jump if Not Equal to 0  
 I 0 1 1 0 1 1 1 0 0 0 0 1 1 CB\2 (R mode form)  
 [ DISPLACEMENT\16 ]

Calculates an effective address, EA. Adds 1 to the contents of the index register, X. If the incremented value does not equal 0, the instruction loads EA into the program counter. If the incremented value is equal to 0, execution continues with the next sequential instruction. Leaves the values of CBIT, LINK, and the condition codes unchanged.

#### Note

This instruction cannot do indexing.

► JMP address  
 Jump  
 I X 0 0 0 1 1 1 0 0 0 Y 0 0 BR\2 (V mode long)  
 DISPLACEMENT\16  
  
 I X 0 0 0 1 1 1 0 0 0 0 0 0 CB\2 (R mode long)  
 [ DISPLACEMENT\16 ]  
  
 I X 0 0 0 1 DISPLACEMENT\10 (S mode; R, V mode short)

Calculates an effective address, EA. Loads EA into the program counter. Leaves the values of CBIT, LINK, and the condition codes unchanged.

► JST address  
 Jump and Store  
 I X 1 0 0 0 1 1 0 0 0 Y 0 0 BR\2 (V mode long)  
 DISPLACEMENT\16

I X 1 0 0 0 1 1 0 0 0 0 0 0 CB\2 (R mode long)  
 [ DISPLACEMENT\16 ]

I X 1 0 0 0 DISPLACEMENT\10 (S mode; R, V mode short)

Calculates an effective address, EA. Stores the contents of the program counter in the location specified by EA. Execution continues at the location EA+1.

The JST instruction truncates the return address according to the addressing mode before storing it. The high-order bits of the memory location are not affected by the store. This allows you to preset the I or X bits in some modes as follows:

<u>Mode</u>	<u>Allowed Presets</u>
16S	I, X
32S, 32R	I
64R, 64V	none

#### Note

JST cannot be used in shared code. In Ring 0, JST inhibits interrupts during execution of the next instruction.

This instruction may call only those subroutines residing in the same procedure segment as the instruction, because only the offset number field of the program counter is saved.

► JSX address  
 Jump and Save in X  
 I 1 1 1 0 1 1 1 0 0 0 Y 1 1 BR\2 (V mode long)  
 DISPLACEMENT\16

I 1 1 1 0 1 1 1 0 0 0 0 1 1 CB\2 (R mode long)  
 [ DISPLACEMENT\16 ]

Calculates an effective address, EA. Increments the contents of the program counter by 1 and loads the result into X. Loads EA into the program counter. For the 750 and 850, if the value of CB is 2 or 3, then the next 16 bits are skipped. Leaves the values of CBIT, LINK, and the condition codes unchanged.

Note

JSX cannot do indexing. See Appendix B for more information.

This instruction may call only those subroutines residing in the same procedure segment as the instruction, because only the offset number field of the program counter is saved.

► JSXB address  
 Jump and Save in XB  
 I X 1 1 0 0 1 1 0 0 0 Y 1 0 BR\2 (V mode long)  
 DISPLACEMENT\16  
  
 I X 1 1 0 0 1 1 0 0 0 0 1 0 CB\2 (R mode long)  
 [ DISPLACEMENT\16 ]

Calculates an effective address, EA. Loads the contents of the program counter into XB. Loads EA into the program counter. Leaves the values of CBIT, LINK, and the condition codes unchanged.

Note

This instruction can make subroutine calls outside the current segment as well as within.

► JSY address  
 Jump and Save in Y  
 I X 1 1 0 0 1 1 0 0 0 Y 0 0 BR\2 (V mode long)  
 DISPLACEMENT\16  
  
 I X 1 1 0 0 DISPLACEMENT\16 (V mode short)

Calculates an effective address, EA. Loads Y with the location number of the program counter. Loads EA into the program counter. Leaves the values of CBIT, LINK, and the condition codes unchanged.

Note

This instruction may call only those subroutines residing in the same procedure segment as the instruction, because only the offset number field of the program counter is saved.

► **LCEQ**  
 Load A on Condition Code EQ  
 1 1 0 0 0 0 1 1 0 1 0 0 0 0 1 1 (V mode form)

If the condition codes reflect an equal to condition, the instruction loads A with a 1. If the condition codes reflect a not equal condition, the instruction loads A with a 0. Leaves the values of CBIT, LINK, and the condition codes unchanged.

► **LOGE**  
 Load A on Condition Code GE  
 1 1 0 0 0 0 1 1 0 1 0 0 0 1 0 0 (V mode form)

If the condition codes reflect a greater than or equal to condition, the instruction loads A with a 1. If the condition codes reflect a less than condition, the instruction loads A with a 0. Leaves the values of CBIT, LINK, and the condition codes unchanged.

► **LOGT**  
 Load A on Condition Code GT  
 1 1 0 0 0 0 1 1 0 1 0 0 0 1 0 1 (V mode form)

If the condition codes reflect a greater than condition, the instruction loads with a 1. If the condition codes reflect a less than or equal to condition, the instruction loads A with a 0. Leaves the values of CBIT, LINK, and the condition codes unchanged.

► **LCLE**  
 Load A on Condition Code LE  
 1 1 0 0 0 0 1 1 0 1 0 0 0 0 0 1 (V mode form)

If the condition codes reflect a less than or equal to condition, the instruction loads A with a 1. If the condition codes reflect a greater than condition, the instruction loads A with a 0. Leaves the values of CBIT, LINK, and the condition codes unchanged.

► **LCLT**  
 Load A on Condition Code LT  
 1 1 0 0 0 0 1 1 0 1 0 0 0 0 0 0 (V mode form)

If the condition codes reflect a less than condition, the instruction loads A with a 1. If the condition codes reflect a greater than or equal to condition, the instruction loads A with a 0. Leaves the values of CBIT, LINK, and the condition codes unchanged.

► **LCNE**  
 Load A on Condition Code NE  
 1 1 0 0 0 0 1 1 0 1 0 0 0 0 1 0 (V mode form)

If the condition codes reflect a not equal condition, the instruction loads A with a 1. If the condition codes reflect an equal condition, the instruction loads A with a 0. Leaves the values of CBIT, LINK, and the condition codes unchanged.

► **LDA address**  
 Load A  
 I X 0 0 1 0 1 1 0 0 0 Y 0 0 BR\2 (V mode long)  
 DISPLACEMENT\16

I X 0 0 1 0 1 1 0 0 0 0 0 0 CB\2 (R mode long)  
 [ DISPLACEMENT\16 ]

I X 0 0 1 0 DISPLACEMENT\10 (S mode; R, V mode short)

Calculates an effective address, EA. Loads the contents of the location specified by EA into A. Leaves the values of CBIT, LINK, and the condition codes unchanged.

► **LDC flr**  
 Load Character  
 0 0 0 0 0 0 1 0 1 1 0 0 FLR 0 1 0 (V mode form)

If the contents of the specified FLR are nonzero, the instruction fetches the single character pointed to by the appropriate FAR and loads it into bits 9 to 16 of A. When the FAR's bit field contains 0, it specifies the left byte (bits 1 to 8) of the 16-bit addressed quantity; when the bit field contains 8, the right byte (bits 9 to 16) is specified. This instruction loads zeros into bits 1 to 8 of A. Updates the contents of the appropriate FAR by 8 so that they point to the next character. Decrements the contents of the specified FLR by 1. Sets the condition codes to NE.

If the contents of the specified FLR are 0, the instruction sets the condition codes to EQ.

The instruction leaves the values of CBIT and LINK unchanged.

#### Note

This instruction uses FAR0 when FLR0 is specified, and FAR1 when FLR1 is specified.

► **LDL address**  
 Load Long  
 I X 0 0 1 0 1 1 0 0 0 Y 1 1 BR\2 (V mode form)  
 DISPLACEMENT\16

Calculates a long (32-bit) effective address, EA. Loads the 32-bit contents of the location specified by EA into L. Leaves the values of CBIT, LINK, and the condition codes unchanged.

► **LDLR address**  
 Load L From Addressed Register  
 I X 0 1 0 1 1 1 0 0 0 Y 0 1 BR\2 (V mode form)  
 DISPLACEMENT\16

Calculates a 32-bit (1-word) effective address, EA. Loads L with the contents of the register file location specified by the offset portion of EA. Bit 2 and bit 12 of the offset portion of EA determine the actions of this instruction:

<u>Bit 2</u>	<u>Bit 12</u>	<u>Action</u>
1*	----	Ignore bit 1 and bits 3 to 9. The offset portion of EA specifies an absolute register number from 0 to '377.
0*	1	Bits 13 to 16 of the offset portion of EA specify one of the registers '20 to '37 in the current register set.
0	0	Bits 13 to 16 of the offset portion of EA specify one of the registers 0 to '17 in the current register set.

\*This is a restricted instruction.

Leaves the values of CBIT and LINK unchanged; the values of the condition codes are indeterminate. See Chapter 9 of the System Architecture Reference Guide for more information on register sets.

► **LDX address**  
 Load X  
 I 1 1 1 0 1 1 1 0 0 0 Y 0 0 BR\2 (V mode long)  
 DISPLACEMENT\16  
  
 I 1 1 1 0 1 1 1 0 0 0 0 0 0 CB\2 (R mode long)  
 [ DISPLACEMENT\16 ]  
  
 I 1 1 1 0 1 DISPLACEMENT\10 (S, R, V mode short form)

Calculates an effective address, EA. Loads X, the index register, with the contents of the location specified by EA. Leaves the values of CBIT, LINK, and the condition codes unchanged. For 750 and 850 processors in R mode only, if CB contains 2 or 3, the first 16 bits of the next instruction will be skipped.

#### Note

LDX cannot specify indexing, though an address calculated in the indirect chain may do so in 16S mode. See Appendix B for more information.

► **LDY address**  
 Load Y  
 I 1 1 1 0 1 1 1 0 0 0 Y 0 1 BR\2 (V mode form)  
 DISPALCEMENT\16

Calculates an effective address, EA. Loads Y with the contents of the location specified by EA. Leaves the values of CBIT, LINK, and the condition codes unchanged.

#### Note

LDY cannot do indexing. See Appendix B for more information.

► **LEQ**  
 Load A on A Equal to 0  
 1 1 0 0 0 0 0 1 0 0 0 0 1 0 1 1 (S, R, V mode form)

If the contents of A are equal to 0, the instruction loads A with a 1. If the contents of A are not equal to 0, the instruction loads A with a 0. Leaves the values of LINK and CBIT unchanged. The condition codes reflect the result of the comparison. (See Appendix A.)

► **LF**  
 Load False  
 1 1 0 0 0 0 0 1 0 0 0 0 1 1 1 0 (S, R, V mode form)

Loads A with a 0. Leaves the values of LINK and CBIT unchanged. The values of the condition codes are indeterminate.

► **LFEQ**  
 Load A on F Equal to 0  
 1 1 0 0 0 0 1 0 0 1 0 0 1 0 1 1 (V mode form)

If the contents of the floating accumulator are equal to 0, the instruction loads A with a 1. If the F contents are not equal to 0, the instruction loads A with a 0. Leaves the values of LINK and CBIT unchanged. The condition codes reflect the result of the comparison. (See Appendix A.) LFEQ works correctly only on normalized or nearly normalized numbers, because it checks the first 32 fraction bits only for equal to zero and less than zero. (See Chapter 6 in the System Architecture Reference Guide.)

► **LFGE**  
 Load A on Floating Accumulator Greater Than or Equal to 0  
 1 1 0 0 0 0 1 0 0 1 0 0 1 1 0 0 (V mode form)

If the contents of the floating accumulator are greater than or equal to 0, the instruction loads A with a 1. If the F contents are less than 0, the instruction loads A with a 0. Leaves the values of LINK and CBIT unchanged. The condition codes reflect the result of the comparison. (See Appendix A.) LFGE works correctly only on normalized or nearly normalized numbers, because it checks the first 32 fraction bits only for equal to zero and less than zero. (See Chapter 6 in the System Architecture Reference Guide.)

► **LFGT**  
 Load A on Floating Accumulator Greater Than 0  
 1 1 0 0 0 0 1 0 0 1 0 0 1 1 0 1 (V mode form)

If the contents of the floating accumulator are greater than 0, the instruction loads A with a 1. If the F contents are less than or equal to 0, the instruction loads A with a 0. Leaves the values of LINK and CBIT unchanged. The condition codes reflect the result of the comparison. (See Appendix A.) LFGT works correctly only on normalized or nearly normalized numbers, because it checks the first 32 fraction bits only for equal to zero and less than zero. (See Chapter 6 in the System Architecture Reference Guide.)

► **LFLE**  
 Load A on Floating Accumulator Less Than or Equal to 0  
 1 1 0 0 0 0 1 0 0 1 0 0 1 0 0 1 (V mode form)

If the contents of the floating accumulator are less than or equal to 0, the instruction loads A with a 1. If the F contents are greater than 0, the instruction loads A with a 0. Leaves the values of LINK and CBIT unchanged. The condition codes reflect the result of the comparison. (See Appendix A.) LFLE works correctly only on normalized or nearly normalized numbers, because it checks the first 32 fraction bits only for equal to zero and less than zero. (See Chapter 6 in the System Architecture Reference Guide.)

► **LFLI flr,data**  
 Load FIR Immediate  
 0 0 0 0 0 0 1 0 1 1 0 0 FIR 0 1 1 (V mode form)  
 INTEGER\16

Loads the 16-bit, unsigned integer contained in bits 17 to 32 (the second halfword) of the instruction into the specified FIR. Clears the upper bits of the FIR. Leaves the values of CBIT, LINK, the condition codes, and the associated FAR unchanged.

► **LFLT**  
 Load A on Floating Accumulator Less Than 0  
 1 1 0 0 0 0 1 0 0 1 0 0 1 0 0 0 (V mode form)

If the contents of the floating accumulator are less than 0, the instruction loads A with a 1. If the F contents are greater than or equal to 0, the instruction loads A with a 0. Leaves the values of LINK and CBIT unchanged. The condition codes reflect the result of the comparison. (See Appendix A.) LFLT works correctly only on normalized or nearly normalized numbers, because it checks the first 32 fraction bits only for equal to zero and less than zero. (See Chapter 6 in the System Architecture Reference Guide.)

► **LFNE**  
 Load A on Floating Accumulator Not Equal to 0  
 1 1 0 0 0 0 1 0 0 1 0 0 1 0 1 0 (V mode form)

If the contents of the floating accumulator are not equal to 0, the instruction loads A with a 1. If the F contents are equal to 0, the instruction loads A with a 0. Leaves the values of LINK and CBIT unchanged. The condition codes reflect the result of the comparison. (See Appendix A.) LFNE works correctly only on normalized or nearly normalized numbers, because it checks the first 32 fraction bits only for equal to zero and less than zero. (See Chapter 6 in the System Architecture Reference Guide.)

## ► LGE

Load A on Greater Than or Equal to 0

1 1 0 0 0 0 0 1 0 0 0 0 1 1 0 0 (S, R, V mode form)

If the contents of A are greater than or equal to 0, the instruction loads A with a 1. If the contents of A are less than 0, the instruction loads A with a 0. Leaves the values of LINK and CBIT unchanged. The condition codes reflect the result of the comparison. (See Appendix A.) This instruction has the same opcode as LLGE.

## ► LGT

Load A on Greater Than 0

1 1 0 0 0 0 0 1 0 0 0 0 1 1 0 1 (S, R, V mode form)

If the A contents are greater than 0, the instruction loads A with 1. If the A contents are less than or equal to 0, the instruction loads A with 0. Leaves the values of LINK and CBIT unchanged. The condition codes reflect the result of the comparison. (See Appendix A.)

## ► LIOT address

Load IOTLB

0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 (V mode form)  
AP\32

Loads a specified IOTLB entry. The following list shows the contents of the LIOT entry and the origin of the information.

<u>Origin</u>	<u>Description</u>
AP in LIOT	Virtual address in I/O segment (calculated from EA).
Page table	Physical address (translation of virtual address) obtained from I/O segment. If the fault bit is set to 1, a page fault occurs.
L register	Target virtual address containing the segment number and page number to be used by procedures accessing this information. This is used to help invalidate the proper locations in the cache. The segment number and low-order 10 bits (offset number in the page) are ignored.

The values of CBIT, LINK, and the condition codes are indeterminate.

Note

LIOT is a restricted instruction.

**LLE**

Load A on A Less Than or Equal to 0

1 1 0 0 0 0 0 1 0 0 0 0 1 0 0 1 (S, R, V mode form)

If the contents of A are less than or equal to 0, the instruction loads A with 1. If the A contents are greater than 0, the instruction loads A with 0. Leaves the values of LINK and CBIT unchanged. The condition codes contain the result of the comparison. (See Appendix A.)

**LLEQ**

Load A on L Equal to 0

1 1 0 0 0 0 1 1 0 1 0 0 1 0 1 1 (V mode form)

If the contents of L are equal to 0, the instruction loads A with a 1. If the contents of L are not equal to 0, the instruction loads A with a 0. Leaves the values of LINK and CBIT unchanged. The condition codes contain the result of the comparison. (See Appendix A.)

**LIGE**

Load A on L Greater Than or Equal to 0

1 1 0 0 0 0 0 1 0 0 0 0 1 1 0 0 (V mode form)

If the contents of L are greater than or equal to 0, the instruction loads A with a 1. If the contents of L are less than 0, the instruction loads A with a 0. Leaves the values of LINK and CBIT unchanged. The condition codes contain the result of the comparison. (See Appendix A.) This instruction has the same op code as LGE.

**LIGT**

Load A on L Greater Than 0

1 1 0 0 0 0 1 1 0 1 0 0 1 1 0 1 (V mode form)

If the L contents are greater than 0, the instruction loads A with 1. If the L contents are less than or equal to 0, the instruction loads A with 0. Leaves the values of LINK and CBIT unchanged. The condition codes contain the result of the comparison. (See Appendix A.)

► **LLL n**  
 Long Left Logical  
 0 1 0 0 0 0 1 0 0 0 N\6 (S, R, V mode form)

Shifts the contents of A and B to the left, bringing zeros into bit 16 of B. Shifts bits out of bit 1 of B into bit 16 of A. CBIT and LINK contain the value of last bit shifted out of A; the values of all other bits shifted out of A are lost. Leaves the values of the condition codes unchanged.

N contains the two's complement of the number of shifts to perform. If N contains 0, the instruction performs 64 shifts.

► **LLLE**  
 Load A on L Less Than or Equal to 0  
 1 1 0 0 0 0 1 1 0 1 0 0 1 0 0 1 (V mode form)

If the contents of L are less than or equal to 0, the instruction loads A with 1. If the L contents are greater than 0, the instruction loads A with 0. Leaves the values of LINK and CBIT unchanged. The condition codes contain the result of the comparison. (See Appendix A.)

► **LLLT**  
 Load A on L Less Than 0  
 1 1 0 0 0 0 0 1 0 0 0 0 1 0 0 0 (V mode form)

If the contents of L are less than 0, the instruction loads A with 1. If the L contents are greater than or equal to 0, the instruction loads A with 0. Leaves the values of LINK and CBIT unchanged. The condition codes contain the result of the comparison. (See Appendix A.) This instruction has the same operation as LLT.

► **LLNE**  
 Load A on L Not Equal to 0  
 1 1 0 0 0 0 1 1 0 1 0 0 1 0 1 0 (V mode form)

If the contents of L are not equal to 0, the instruction loads A with a 1. If the contents of L are equal to 0, the instruction loads A with a 0. Leaves the values of LINK and CBIT unchanged. The condition codes contain the result of the comparison. (See Appendix A.)

► **LLR n**  
 Long Left Rotate  
 0 1 0 0 0 0 1 0 1 0 N\6 (S, R, V mode form)

Shifts the contents of A and B left, rotating bit 1 of A into bit 16 of B. Bit 1 of B shifts into bit 16 of A. CBIT and LINK contain a copy of the last bit rotated into bit 16 of B. Leaves the values of the condition codes unchanged.

N contains the two's complement of the number of shifts to perform. If N contains 0, the instruction performs 64 shifts.

► **LLS n**  
 Long Left Shift  
 0 1 0 0 0 0 1 0 0 1 N\6 (V mode form)

Shifts the 32-bit integer in L left arithmetically, bringing zeros into bit 32. Bits shifted out of bit 1 are lost. If bit 1 changes state, it is interpreted as an overflow and causes an integer exception. If no integer exception occurs, CBIT is reset to 0. The values of LINK and the condition codes are indeterminate.

If an integer exception occurs and bit 8 of the keys contains 0, the instruction sets CBIT to 1. If bit 8 contains a 1, the instruction sets CBIT to 1 and causes an integer exception fault. See Chapter 10 of the System Architecture Reference Guide for more information.

► **LLS n**  
 Long Left Shift  
 0 1 0 0 0 0 1 0 0 1 N\6 (S, R mode form)

Shifts the 31-bit integer contained in A and B left arithmetically, bringing zeros into bit 16 of B. Bit 1 of B does not take part in the shift; bit 2 of B is shifted into bit 16 of A. Bits shifted out of bit 1 of A are lost. If bit 1 of A changes state, it is interpreted as an overflow and causes an integer exception. If no integer exception occurs, CBIT is reset to 0. The values of LINK and the condition codes are indeterminate.

If an integer exception occurs and bit 8 of the keys contains 0, the instruction sets CBIT to 1. If bit 8 contains a 1, the instruction sets CBIT to 1 and causes an integer exception fault. See Chapter 10 of the System Architecture Reference Guide for more information.

► **LLT**  
 Load on A Less Than 0  
 1 1 0 0 0 0 0 1 0 0 0 0 1 0 0 0 (S, R, V mode form)

If the contents of A are less than 0, the instruction loads A with 1. If the A contents are greater than or equal to 0, the instruction loads A with 0. Leaves the values of LINK and CBIT unchanged. The condition codes contain the result of the comparison. (See Appendix A.) This instruction has the same operation as LLLT.

► **LNE**  
 Load on A Not Equal to 0  
 1 1 0 0 0 0 0 1 0 0 0 0 1 0 1 0 (S, R, V mode form)

If the contents of A are not equal to 0, the instruction loads A with a 1. If the contents of A are equal to 0, the instruction loads A with a 0. Leaves the values of LINK and CBIT unchanged. The condition codes contain the result of the comparison. (See Appendix A.)

► **LPID**  
 Load Process ID  
 0 0 0 0 0 0 0 1 1 0 0 0 1 1 1 1 (V mode form)

Loads the process ID from bits 1 to 10 of A into RPID (the process ID register). This contains the 10 most significant bits of the user's address space. Leaves the values of CBIT, LINK, and the condition codes unchanged.

The RPID data is used to update the process ID field of an STLB entry as required. This RPID data is later used during subsequent memory accesses to verify that STLB data is still valid (STLB hit) or not (STLB miss). This register is for internal machine operation, and should not normally be modified by the user.

#### Note

This is a restricted instruction.

► **LPSW address**  
 Load PSW  
 0 0 0 0 0 0 0 1 1 1 0 0 1 0 0 1 (V mode form)  
 AP\32

Changes the status of the processor by loading new values into the program counter, keys, and modals. Inhibits interrupts for one instruction.

Addresses a 64-bit (4-halfword) block at the specified location. The block has the following.

<u>Offset in Block</u>	<u>Contents</u>
1 to 2	New program counter (ring, segment, offset numbers)
3	New keys
4	New modals

LPSW loads the program counter and keys of the currently running process with the contents of the first three offsets (bits 1 to 48), then loads the processor modals with the contents of the fourth offset (bits 49 to 64).

The new value of bit 15 in the keys, the in-dispatch bit, can temporarily halt execution of the current process. This bit is altered by software only during a cold or warm start. If bit 15 is 0, the currently executing process will continue to execute, but at a location defined by the new value of the program counter. If bit 15 is 1, the processor enters the dispatcher and dispatches the ready process with the highest priority. When execution resumes for the process that was temporarily halted, note that execution resumes at the point defined by the value of the new program counter.

Regardless of the value of bit 15, the new value of the modals takes effect immediately, since the modals are associated with the processor, not the process.

This instruction loads the 64 bits (four halfwords) of the register set that the STLR instruction cannot correctly load. STLR does not update the separate hardware registers the processor uses to maintain duplicate information for optimization.

Never use this instruction to change bits 9 to 11 of the modals. These bits specify the current user register set. This means that if you do not know the current value of these bits, you must do the following each time you want to execute an LPSW.

1. Inhibit interrupts.
2. Read the current values of modal bits 9 to 11 (use LDLR).
3. Mask the old values of the modal bits into the new information.
4. Load the new information into the modals with an LPSW.

For the two common uses of LPSW, you do not have to perform this sequence, since the values of modal bits 9 to 11 are predictable. When you use LPSW after a Master Clear to turn on processor exchange mode,

bits 9 to 11 are 010 because the processor is always initialized to register set 2. When you use LPSW to return from a fault, check, or interrupt, simply reload the values stored by the break because these values are still correct.

Also note that you should not use LPSW to set bits 16 (the save done bit) or 15 (the in-dispatcher bit) of the keys, unless you are merely loading status following a fault, check, or interrupt. When issuing LPSW after a Master Clear, make sure you load zeros into both of these bits.

### Note

LPSW is a restricted instruction. This instruction inhibits interrupts during execution of the next instruction.

► **IRL n**  
Long Right Logical  
0 1 0 0 0 0 0 0 0 0 N\6 (S, R, V mode form)

Shifts the contents of A and B right, bringing zeros into bit 1 of A. Shifts bit 16 of A into bit 1 of B. CBIT and LINK contain the value of the last bit shifted out of B; the values of all other bits shifted out of B are lost. Leaves the values of the condition codes unchanged.

N contains the two's complement of the number of shifts to perform. If N contains 0, the instruction performs 64 shifts.

► **IIR n**  
Long Right Rotate  
0 1 0 0 0 0 0 0 1 0 N\6 (S, R, V mode form)

Shifts the contents of A and B right, rotating bit 16 of B into bit 1 of A. Shifts bit 16 of A into bit 1 of B. CBIT and LINK contain a copy of the last bit rotated from B to A. Leaves the values of the condition codes unchanged.

N contains the two's complement of the number of shifts to perform. If N contains 0, the instruction performs 64 shifts.

► **IRS n**  
Long Right Shift  
0 1 0 0 0 0 0 0 0 1 N\6 (V mode form)

Shifts the 32-bit integer contained in L right arithmetically. Shifts copies of bit 1, the sign bit, into each of the vacated bits. CBIT and LINK contain the value of the last bit shifted out of L; the values of

## INSTRUCTION SETS GUIDE

all other bits shifted out are lost. Leaves the values of the condition codes unchanged.

N contains the two's complement of the number of shifts to perform. If N contains 0, the instruction performs 64 shifts.

► LRS n  
Long Right Shift  
0 1 0 0 0 0 0 0 0 1 N\6 (S, R mode form)

Shifts right arithmetically the 31-bit integer contained in A and B, leaving bit 1 of A unaffected. Bit 1 of B does not take part in the shift; bit 16 of A is shifted into bit 2 of B. Shifts copies of bit 1 of A into each of the vacated bits. CBIT and LINK contain the value of the last bit shifted out of B; the values of all other bits shifted out of B are lost. Leaves the values of the condition codes unchanged.

N contains the two's complement of the number of shifts to perform. If N contains 0, the instruction performs 64 shifts.

► LT  
Load True  
1 1 0 0 0 0 0 1 0 0 0 0 1 1 1 1 (S, R, V mode form)

Loads A with a 1. Leaves the values of LINK and CBIT unchanged. The values of the condition codes are indeterminate.

► MPL address  
 Multiply Long  
 I X 1 1 1 0 1 1 0 0 0 Y 1 1 BR\2 (V mode form)  
 DISPLACEMENT\16

Calculates an effective address, EA. Multiplies the 32-bit integer in L by the 32-bit integer in the location specified by EA. Stores the 64-bit result in L and E. The 150/250, 450/550/250-II, I450-II, and 2250 processors leave the CBIT and LINK unchanged. The other 50 Series processors reset the CBIT to 0 and leave the value of LINK indeterminate. For all 50 Series processors, the condition codes are unchanged. MPL cannot cause overflow or generate an integer exception.

► MPY address  
 Multiply  
 I X 1 1 1 0 1 1 0 0 0 Y 0 0 BR\2 (V mode long)  
 DISPLACEMENT\16

I X 1 1 1 0 DISPLACEMENT\10 (V mode short)

Calculates an effective address, EA. Multiplies the 16-bit integer in A by the 16-bit integer in the location specified by EA. Stores the 32-bit result in A and B. Resets the CBIT to 0. The value of LINK is indeterminate. Leaves the values of the condition codes unchanged.

#### Note

This instruction cannot cause overflow.

► MPY address  
 Multiply  
 I X 1 1 1 0 1 1 0 0 0 0 0 0 CB\2 (R mode long)  
 [ DISPLACEMENT\16 ]

I X 1 1 1 0 DISPLACEMENT\10 (S mode; R mode short)

Calculates an effective address, EA. Multiplies the 16-bit integer in A by the 16-bit integer in the location specified by EA. Loads the 31-bit result in A and B. If the multiplier and multiplicand are both  $-(2^{*15})$ , an integer exception occurs. If no integer exception occurs, CBIT is reset to 0. The value of LINK is indeterminate. For the 2350 to 9955 II, the condition codes are unchanged. For the earlier processors listed in "About This Book", the values of the condition codes reflect the result of the operation. (See Appendix A.)

If an integer exception occurs and bit 8 of the keys contains 0, the instruction sets CBIT to 1. If bit 8 contains a 1, the instruction sets CBIT to 1 and causes an integer exception fault. See Chapter 10 of the System Architecture Reference Guide for more information.

► NFYB address  
 Notify to Beginning  
 0 0 0 0 0 0 1 0 1 0 0 0 1 0 0 1 (V mode form)  
 AP\32

Notifies the semaphore at the address specified by the address pointer in the instruction. Uses LIFO (last in, first out) queueing. Does not clear the currently active interrupt. The values of CBIT, LINK, and the condition codes are indeterminate. See Chapter 9 of the System Architecture Reference Guide for more information.

#### Note

This is a restricted instruction.

► NFYE address  
 Notify to End  
 0 0 0 0 0 0 1 0 1 0 0 0 1 0 0 0 (V mode form)  
 AP\32

Notifies the semaphore at the address specified by the address pointer in the instruction. Uses FIFO (first in, first out) queueing. Does not clear the currently active interrupt. The values of CBIT, LINK, and the condition codes are indeterminate. See Chapter 9 of the System Architecture Reference Guide for more information.

#### Note

This is a restricted instruction.

► NOP  
 No Operation  
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 (S, R, V mode form)

Does nothing. Leaves the values of CBIT, LINK, and the condition codes unchanged.

► OCP function,device  
 Output Control Pulse  
 0 0 1 1 0 0 FUNCTION\4 DEVICE\6 (S, R mode form)

Sends a control pulse to perform the specified function to the specified device. This instruction never skips. Leaves the values of CBIT, LINK, and the condition codes unchanged. See Chapter 11 of the System Architecture Reference Guide for more information.

#### Note

This is a restricted instruction.

► ORA address  
 Inclusive OR  
 I X 0 0 1 1 1 1 0 0 0 Y 1 0 BR\2 (V mode form)  
 DISPLACEMENT\16

Calculates an effective address, EA. Logically ORs the contents of the location specified by EA and the contents of A and stores the result in A. Leaves the values of CBIT, LINK, and the condition codes unchanged.

► OTA function,device  
 Output From A  
 1 1 1 1 0 0 FUNCTION\4 DEVICE\6 (S, R mode form)

Transfers data from A to the specified device. Leaves the values of CBIT, LINK, and the condition codes unchanged. See Chapter 11 of the System Architecture Reference Guide for more information.

#### Note

This is a restricted instruction.

► OTK  
 Output Keys  
 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 1 (S, R mode form)

Stores the contents of A in the keys. Loads CBIT, LINK, and the condition codes as a result of the operation. Loads the low-order 8 bits of the floating exponent (address trap location 6) register with the low-order 8 bits of A. If this instruction is executed in Ring 0, it inhibits interrupt during execution of the next instruction.

► PCL address  
 Procedure Call  
 I X 1 0 0 0 1 1 0 0 0 Y 1 0 BR\2 (V mode form)  
 DISPLACEMENT\16

Sets CBIT, LINK, and the condition codes to the values contained in the ECB. See Chapter 8 of the System Architecture Reference Guide for a complete description of this instruction.

#### Note

When arguments are to be transferred to the called procedure, this instruction uses X and Y, destroying the previous contents of these registers. XB is updated if an AP has the S bit = 0. The contents of X, Y, and XB remain unchanged if no arguments are transferred. The contents of the condition codes, CBIT, and LINK are not correctly saved in the ECB along with the rest of the caller's keys.

► PID  
 Position for Integer Divide  
 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 1 (S, R mode form)

Moves the contents of bits 2 to 16 of A into bits 2 to 16 of B. Clears bit 1 of register B to 0 and extends the sign contained in bit 1 of A into bits 2 to 16 of A. Leaves the values of CBIT, LINK, and the condition codes unchanged.

► PIDA  
 Position for Integer Divide  
 0 0 0 0 0 0 0 0 0 0 1 0 0 1 1 0 1 (V mode form)

Moves the contents of bits 1 to 16 of A into bits 17 to 32 of L. Extends the sign contained in bit 1 of A into bits 2 to 16 of A. Leaves the values of CBIT, LINK, and the condition codes unchanged.

► PIDL  
 Position for Integer Divide Long  
 0 0 0 0 0 0 0 0 0 1 1 0 0 0 1 0 1 (V mode form)

Moves the contents of L into E and extends the sign contained in bit 1 of L into bits 2 to 32 of L. Leaves the values of CBIT, LINK, and the condition codes unchanged.

## ► PIM

Position After Multiply

0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 1 (S, R mode form)

Moves bits 2 to 16 of B into bits 2 to 16 of A. This converts a 31-bit integer to a 16-bit integer. Leaves the values of CBIT, LINK, and the condition codes unchanged. Overflow does not cause an integer exception.

## ► PIMA

Position After Multiply

0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 (V mode form)

Moves bits 17 to 32 of L into bits 1 to 16 of A. This converts a 32-bit integer to a 16-bit integer. An integer exception occurs if there is an overflow. (This occurs if bits 1 to 17 of L contain a value other than all zeros or all ones before the move.) If no integer exception occurs, CBIT is reset to 0. The values of LINK and the condition codes are indeterminate.

If an integer exception occurs and bit 8 of the keys contains 0, the instruction sets CBIT to 1. If bit 8 contains a 1, the instruction sets CBIT to 1 and causes an integer exception fault. See Chapter 10 of the System Architecture Reference Guide for more information.

Note

To position bits 17 to 32 of L in A, PIMA can modify all 32 bits of L. Since A and B overlap L, this swap means that the contents of B are indeterminate at the end of this instruction.

## ► PIML

Position After Integer Multiply Long

0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 1 (V mode form)

Moves the contents of bits 1 to 32 of E into bits 1 to 32 of L. This converts a 64-bit integer to a 32-bit integer. An overflow causes an integer exception. If no integer exception occurs, CBIT is reset to 0. The values of LINK and the condition codes are indeterminate.

If an integer exception occurs and bit 8 of the keys contains 0, the instruction sets CBIT to 1. If bit 8 contains a 1, the instruction sets CBIT to 1 and causes an integer exception fault. See Chapter 10 of the System Architecture Reference Guide for more information.

► **PRTN**  
 Procedure Return  
 0 0 0 0 0 0 0 1 1 0 0 0 1 0 0 1 (V mode form)

Deallocates the stack frame created for the executing procedure and returns to the environment of the procedure that called it.

To deallocate the frame, the instruction stores the current value of the stack base register into the free pointer. It then restores the caller's state by loading the caller's program counter, stack base register, linkage base register, and keys with the values contained in the frame being deallocated. Sets bits 15 to 16 of the keys to 0.

Loads the ring number in the program counter with the current ring number to allow outward returns but prevent inward returns.

► **PTLB**  
 Purge TLB  
 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 0 0 (V mode form)

L contains the address of a physical page, right justified. Based on the value of L bit 1, PTLB purges either the first 128 locations or a single location. If L bit 1 contains a 1, the instruction performs a complete purge. If L bit 1 contains a 0, the instruction purges the page specified by L. Leaves the values of CBIT, LINK, and the condition codes indeterminate. See Chapters 1, 4, and 11 of the System Architecture Reference Guide for more information about the STLB and IOTLB.

#### Note

This is a restricted instruction.

On the 750, 850, and 2350 to 9955 II, insert a CRE (Clear E) instruction before PTLB. Since PTLB uses E as a pointer, the CRE zeros E before PTLB manipulates it. If an interrupt occurs during PTLB's execution, E points to the location PTLB is currently purging. PTLB leaves the contents of E in an undefined state at the end of its execution.

► QFAD address  
 Quad Precision Floating Add  
 I X 0 1 0 1 1 1 0 0 0 Y 1 0 BR\2 (V mode long)  
 DISPLACEMENT\16  
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0

Calculates an effective address, EA. Adds the 112-bit, quad precision number contained in the locations specified by EA to the contents of QAC. (See Chapter 6 of the System Architecture Reference Guide.) Normalizes the result and loads it into QAC. An overflow or underflow causes a floating-point exception. If no floating-point exception occurs, the instruction resets CBIT to 0. The values of LINK and the condition codes are indeterminate.

If a floating-point exception occurs and bit 7 of the keys contains a 1, the instruction sets CBIT to 1. If bit 7 contains a 0, the instruction sets CBIT to 1 and causes a floating-point exception fault. See Chapter 10 of the System Architecture Reference Guide for more information.

#### Note

If QFAD is used for any earlier system listed in "About This Book", an unimplemented instruction (UII) fault occurs. (See Chapter 10 of the System Architecture Reference Guide.)

► QFCM  
 Quad Precision Floating Complement  
 1 1 0 0 0 0 0 1 0 1 1 1 1 0 0 0 (V mode form)

Forms the two's complement of the value contained in QAC and normalizes it if necessary. (See Chapter 6 of the System Architecture Reference Guide.) Stores the result in QAC. An underflow or overflow causes a floating-point exception. If no floating-point exception occurs, resets CBIT to 0. The values of LINK and the condition codes are indeterminate.

If a floating-point exception occurs and bit 7 of the keys contains a 1, the instruction sets CBIT to 1. If bit 7 contains a 0, the instruction sets CBIT to 1 and causes a floating-point exception fault. See Chapter 10 of the System Architecture Reference Guide for more information.

#### Note

If QFCM is used for any earlier system listed in "About This Book", an unimplemented instruction (UII) fault occurs. (See Chapter 10 of the System Architecture Reference Guide.)

► QFCS address  
 Quad Precision Floating Point Compare and Skip  
 I X 0 1 0 1 1 0 0 0 Y 1 0 ER\2 (V mode long)  
 DISPLACEMENT\16  
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0

Calculates an effective address, EA. Compares the contents of QAC (see Chapter 6 of the System Architecture Reference Guide) to the 112-bit contents of the location specified by EA and skips as shown below.

<u>Condition</u>	<u>Skip</u>
QAC > EA contents.	No skip.
QAC = EA contents.	Skip 16 bits (one halfword).
QAC < EA contents.	Skip 32 bits (two halfwords).

The values of CBIT, LINK, and the condition codes are indeterminate. On some processors, QFCS works correctly only on normalized numbers as follows. The comparison has a maximum of three sequential stages: first the signs, then the exponents, and finally the fractions of the two numbers are compared for equality. If the comparison during any one of these stages reveals an inequality, the results are returned and the instruction ends. Unnormalized numbers are unexpected and produce unexpected results. Other processors actually perform a subtract, resulting in a proper comparison.

#### Note

If QFCS is used for any earlier system listed in "About This Book", an unimplemented instruction (UII) fault occurs. (See Chapter 10 of the System Architecture Reference Guide.)

► QFDV address  
 Quad Precision Floating Point Divide  
 I X 0 1 0 1 1 1 0 0 0 Y 1 0 ER\2 (V mode long)  
 DISPLACEMENT\16  
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1

Calculates an effective address, EA. Divides the contents of QAC by the 112-bit contents of the location specified by EA. Normalizes the result and stores the whole quotient into QAC. An overflow, underflow, or divide by 0 causes a floating-point exception. If there is no floating-point exception, resets CBIT to 0. The values of LINK and the condition codes are indeterminate.

If a floating-point exception occurs and bit 7 of the keys contains a 1, the instruction sets CBIT to 1. If bit 7 contains a 0, the instruction sets CBIT to 1 and causes a floating-point exception fault. See Chapter 10 of the System Architecture Reference Guide.

Note

If QFDV is used for any earlier system listed in "About This Book", an unimplemented instruction (UII) fault occurs. (See Chapter 10 of the System Architecture Reference Guide.)

► QFLD address  
 Quad Precision Floating Point Load  
 I X 0 1 0 1 1 1 0 0 0 Y 1 0 BR\2 (V mode long)  
 DISPLACEMENT\16  
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Calculates an extended, augmented effective address, EA. Performs one of the following actions with the value contained in the location specified by EA. Loads bits 1 to 112 into QAC and zeros QAC bits 113 to 128, or loads 128 bits into QAC. In either case, no normalization occurs. (See Chapter 6 of the System Architecture Reference Guide for more information.) Leaves the values of CBIT, LINK, and the condition codes unchanged.

Note

If QFLD is used for any earlier system listed in "About This Book", an unimplemented instruction (UII) fault occurs. (See Chapter 10 of the System Architecture Reference Guide.)

► QFLX address  
 Quad Precision Floating Point Load Index  
 I 0 1 1 0 1 1 1 0 0 0 Y 1 1 BR\2 (V mode long)  
 DISPLACEMENT\16

Calculates an effective address, EA. Shifts the 16-bit contents of the location specified by EA to the left three times to multiply the contents by eight. Shifts in zeros on the right and shifts data out on the left first through bit 2 and then bit 1. Leaves the values of CBIT, LINK, and the condition codes unchanged.

Note

QFLX cannot do indexing. See Appendix B for more information.

If QFLX is used for any earlier system listed in "About This Book", an unimplemented instruction (UII) fault occurs. (See Chapter 10 of the System Architecture Reference Guide.)

► QFMP address  
 Quad Precision Floating Point Multiply  
 I X 0 1 0 1 1 1 0 0 0 Y 1 0 ER\2 (V mode long)  
 DISPLACEMENT\16  
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0

Calculates an effective address, EA. Multiplies the contents of QAC by the 112-bit contents of the location specified by EA. (See Chapter 6 of the System Architecture Reference Guide.) Normalizes the result if necessary and stores it into QAC. An overflow or underflow causes a floating-point exception. If there is no floating-point exception, the instruction resets CBIT to 0. The values of LINK and the condition codes are indeterminate.

If a floating-point exception occurs and bit 7 of the keys contains a 1, the instruction sets CBIT to 1. If bit 7 contains a 0, the instruction sets CBIT to 1 and causes a floating-point exception fault. See Chapter 10 of the System Architecture Reference Guide for more information.

#### Note

If QFMP is used for any earlier system listed in "About This Book", an unimplemented instruction (UII) fault occurs. (See Chapter 10 of the System Architecture Reference Guide.)

► QFSB address  
 Quad Precision Floating Point Subtract  
 I X 0 1 0 1 1 1 0 0 0 Y 1 0 ER\2 (V mode long)  
 DISPLACEMENT\16  
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1

Calculates an effective address, EA. Subtracts the contents of the locations specified by EA from the 112-bit contents of QAC. (See Chapter 6 of the System Architecture Reference Guide.) Normalizes the result if necessary and loads it into QAC. An overflow or underflow causes a floating-point exception. If there is no floating-point exception, the instruction resets CBIT to 0. The values of LINK and the condition codes are indeterminate.

If a floating-point exception occurs and bit 7 of the keys contains a 1, the instruction sets CBIT to 1. If bit 7 contains a 0, the instruction sets CBIT to 1 and causes a floating-point exception fault. See Chapter 10 of the System Architecture Reference Guide for more information.

Note

If QFSB is used for any earlier system listed in "About This Book", an unimplemented instruction (UII) fault occurs. (See Chapter 10 of the System Architecture Reference Guide.)

► QFST address  
 Quad Precision Floating Point Store  
 I X 0 1 0 1 1 1 0 0 0 Y 1 0 BR\2 (V mode long)  
 DISPLACEMENT\16  
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1

Calculates an effective address, EA. Stores the 128-bit contents of QAC into the 128 bits of memory specified by EA. (See Chapter 6 of the System Architecture Reference Guide.) Leaves the values of CBIT, LINK, and the condition codes unchanged.

Note

This instruction does not normalize the result before storing it into the specified memory location.

If QFST is used for any earlier system listed in "About This Book", an unimplemented instruction (UII) fault occurs. (See Chapter 10 of the System Architecture Reference Guide.)

► QINQ  
 Quad to Integer, in Quad Convert  
 1 1 0 0 0 0 0 1 0 1 1 1 1 0 1 0 (V mode form)

Strips the fractional portion of QAC as described in Table 2-4.

Table 2-4  
QINQ Actions

Exponent Value	Action
'337 <= Exp	No operation.
'200 < Exp < '337	If sign >= 0, strip fractional part of QAC for result. If sign < 0 and fractional part <> 0, strip fractional part of QAC and increment integer portion of QAC by 1. If sign < 0 and fractional part = 0, no action is done.
'200 = Exp	If sign >= 0, result = 0. If sign < 0 and bits 2 to 96 = 0, result = -1. If sign < 0 and bits 2 to 96 <> 0, result = 0.
'200 > Exp	Result = 0.

The QINQ instruction can cause a floating-point exception; an exception does not alter the contents of QAC. If no floating-point exception occurs, the instruction resets CBIT to 0. The values of LINK and the condition codes are indeterminate.

If a floating-point exception occurs and bit 7 of the keys contains a 1, the instruction sets CBIT to 1. If bit 7 contains a 0, the instruction sets CBIT to 1 and causes a floating-point exception fault. See Chapter 10 of the System Architecture Reference Guide for more information.

#### Note

If QINQ is used for any earlier system listed in "About This Book", an unimplemented instruction (UII) fault occurs. (See Chapter 10 of the System Architecture Reference Guide.)

► QIQR  
 Quad to Integer, in Quad Convert Rounded  
 1 1 0 0 0 0 0 1 0 1 1 1 1 0 1 1 (V mode form)

Strips the fractional portion of QAC as described in Table 2-5.

Table 2-5  
QIQR Actions

Exponent Value	Action
'337 <= Exp	No operation.
'177 < Exp < '337	If sign >= 0, round.* If sign < 0 and fractional part <> 0.5,** round and strip the fractional part of QAC.
Exp = '177	If sign >= 0, result = 0. If sign < 0 and bits 2 to 96 = 0, result = -1. If sign < 0 and bits 2 to 96 <> 0, result = 0. For all cases increment integer part by 1 if it exists and the most significant bit of QAC = 1.
Exp < '177	The result is 0.

\* Rounding occurs if the MSB of the QAC fraction is 1. For example, add the MSB of the QAC fraction to itself and carry out to the QAC integer.

\*\* 0.5 implies a QAC fraction with the MSB = 1 and all other bits = 0.

The QIQR instruction can cause a floating-point exception; an exception does not alter the contents of QAC. If no floating-point exception occurs, the instruction resets CBIT to 0. The values of LINK and the condition codes are indeterminate.

If a floating-point exception occurs and bit 7 of the keys contains a 1, the instruction sets CBIT to 1. If bit 7 contains a 0, the instruction sets CBIT to 1 and causes a floating-point exception fault. See Chapter 10 of the System Architecture Reference Guide for more information.

#### Note

If QIQR is used for any earlier system listed in "About This Book", an unimplemented instruction (UII) fault occurs. (See Chapter 10 of the System Architecture Reference Guide.)

► **RBQ address**  
 Remove Entry From Bottom of Queue  
 1 1 0 0 0 0 1 1 1 1 0 0 1 1 0 1 (V mode form)  
 AP\32

The address pointer in this instruction points to the QCB for a queue. The instruction removes the entry from the bottom of the referenced queue and loads it into A. If the queue is not empty, sets the condition codes to NE; if empty, resets A to 0 and sets the condition codes to EQ. Leaves the values of CBIT and LINK unchanged.

► **RCB**  
 Reset CBIT to 0  
 1 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 (S, R, V mode form)

Resets CBIT to 0. Leaves the values of LINK and the condition codes unchanged.

► **RMC**  
 Reset Machine Check Flag to 0  
 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 (S, R, V mode form)

Resets the MCM flag (bits 15 to 16 of the modals) to 0. Leaves the values of CBIT, LINK, and the condition codes unchanged. Inhibits interrupts during execution of the next instruction.

#### Note

This is a restricted instruction.

► **RRST address**  
 Restore Registers  
 0 0 0 0 0 0 0 1 1 1 1 0 0 1 1 1 1 (V mode form)  
 AP\32

Calculates an effective address, EA, from the 32-bit address pointer in the instruction. This specifies the starting address of a save area for the general, floating, and XB registers. The save area format is shown in Table 2-6. Restores the contents of the general, floating, and XB registers from this save area. Bits 1 to 16 of the save area are a save mask, whose format appears in Figure 2-4. A mask bit value of 1 means that the corresponding register had nonzero contents that have been saved in the save area; a mask bit value of 0 means that the corresponding register's contents were 0. Leaves the values of CBIT, LINK, and the condition codes unchanged.

Table 2-6  
RRST Save Area Format

Offset #	Contents
1	Save mask
2 to 5	FR1 (F)
6 to 9	FRO
10 to 11	X, GR7
12 to 13	GR6
14 to 15	Y, S, GR5
15 to 17	GR4
18 to 19	E, GR3
20 to 21	A, B, L, GR2
22 to 23	GR1
24 to 25	GR0
26 to 27	XB

1	4	5	6	7	8	9	10	11	12	13	14	15	16
0000	FR1	FRO	X	-	Y	-	E	L,B,A	---				

Save Mask Format, RRST and RSAV Instructions  
Figure 2-4

► RSAV address  
Save Registers  
0 0 0 0 0 0 0 1 1 1 0 0 1 1 0 1 (V mode form)  
AP\32

Calculates an effective address, EA, from the 32-bit address pointer in the instruction. This specifies the starting address of a save area for the general, floating, and XB registers. The save area format is shown in Table 2-7. Bits 1 to 16 of the save area are a save mask, whose format appears in Figure 2-5. This instruction sets the mask bit of each register as follows: to 1 if the register's contents have a nonzero value; to 0 if a 0 value. Saves the nonzero contents of the general, floating, and XB registers in the save area. Leaves the values of CBIT, LINK, and the condition codes unchanged.

Table 2-7  
RSAV Save Area Format

Offset #	Contents
1	Save mask
2 to 5	FR1 (F)
6 to 9	FRO
10 to 11	X, GR7
12 to 13	GR6
14 to 15	Y, S, GR5
16 to 17	GR4
18 to 19	E, GR3
20 to 21	A, B, L, GR2
22 to 23	GR1
24 to 25	GRO
26 to 27	XB

1	4	5	6	7	8	9	10	11	12	13	14	15	16
0000	FR1	FRO	X	-	Y	-	E	L,B,A	---				

Save Mask Format, RRSF and RSAV Instructions  
Figure 2-5

► RTQ address  
Remove Entry From Top of Queue  
1 1 0 0 0 0 1 1 1 1 0 0 1 1 0 0 (V mode form)  
AP\32

The address pointer in this instruction is to the QCB for a queue. The instruction removes the entry from the top of the referenced queue, and loads it into A. If the queue is empty, the instruction resets A to 0 and the condition codes to EQ; if not empty, sets the condition codes to NE. Leaves the values of CBIT and LINK unchanged.

► RTS  
Reset Time Slice  
0 0 0 0 0 0 0 1 0 1 0 0 1 0 0 1 (V mode form)  
Valid for the 550-II, 750, 850, I450, and new processors.

The A register contains a negative value representing the number of milliseconds in the new time slice. The time slice is determined by counting ITH up every 1.024 milliseconds until zero, when the time

slice ends. Therefore, ITH is the two's complement of the number of milliseconds remaining in the time slice. The elapsed timer contains the total number of 1.024 millisecond units that have elapsed since process creation plus the full count of the current time slice. Combining ITH and ET by addition gives the total elapsed time.

RTS adds the current value of the interval timer (locations 16 to 17 of the PCB) to the contents of the elapsed timer (locations 10 to 11 of the PCB), then subtracts the contents of A from the sum of the timers. Stores the result in the elapsed timer. Loads the contents of A into the interval timer. Leaves the contents of A unchanged. The values of CBIT, LINK, and the condition codes are unchanged.

The addition performed by this instruction is equivalent to the following series of instructions.

```

LDA  ITH  /* load A with the contents of ITH
SUB   RV  /* subtract reset value (in RV) from contents of A
PIDA           /* sign extend the contents of A into L bits 17 to 32
SRC           /* skip next 16-bit halfword if CBIT is 0 (no overflow)
CMA           /* complement A
ADL  ET    /* add contents of L and contents of ET
STL  ET    /* store contents of L in ET
LDA  RV     /* load A with reset value
STA  ITH    /* store the reset value into ITH

```

#### Note

RTS is a restricted instruction.

**S1A**

Subtract 1 From A

1 1 0 0 0 0 0 0 0 1 0 0 1 0 0 0 (S, R, V mode form)

Subtracts 1 from the contents of A and stores the result in A. If the number to be decremented is  $-(2^{15})$ , an integer exception occurs, and the instruction loads  $(2^{15})-1$  into A. If no overflow occurs, the instruction resets CBIT to 0. LINK contains the borrow bit. The condition codes reflect the result of the operation. (See Appendix A.)

If an integer exception occurs and bit 8 of the keys contains 0, the instruction sets CBIT to 1. If bit 8 contains a 1, the instruction sets CBIT to 1 and causes an integer exception fault. See Chapter 10 of the System Architecture Reference Guide for more information.

**S2A**

Subtract 2 From A

1 1 0 0 0 0 0 0 1 1 0 0 1 0 0 0 (S, R, V mode form)

Subtracts 2 from the contents of A and stores the result in A. If the number to be decremented is  $-(2^{15})-1$  or  $-2^{15}$ , an integer exception occurs and the instruction loads  $(2^{15})-1$  or  $(2^{15})-2$ , respectively, into A. If no overflow occurs, the instruction resets CBIT to 0. LINK contains the borrow bit. The condition codes reflect the result of the operation. (See Appendix A.)

If an integer exception occurs and bit 8 of the keys contains 0, the instruction sets CBIT to 1. If bit 8 contains a 1, the instruction sets CBIT to 1 and causes an integer exception fault. See Chapter 10 of the System Architecture Reference Guide for more information.

**SAR n**

Skip on A Register Bit Reset to 0

1 0 0 0 0 0 0 0 1 0 1 1 N\4 (S, R, V mode form)

Skips the next 16-bit halfword if bit n in register A contains 0. Leaves the values of CBIT, LINK, and the condition codes unchanged.

N specifies the bit to test. A value of 0 indicates bit 1; 1, bit 2; and so on.

Note

The assembler converts n to the octal equivalent of bit number minus 1.

► SAS n  
 Skip on A Register Bit Set to 1  
 1 0 0 0 0 0 1 0 1 0 1 1 N\4 (S, R, V mode form)

Skips the next 16-bit halfword if bit n in register A contains 1.  
 Leaves the values of CBIT, LINK, and the condition codes unchanged.

N specifies the bit to test. A value of 0 indicates bit 1, and so on.

#### Note

The assembler converts n to the octal equivalent of bit number minus 1.

► SBL address  
 Subtract Long  
 I X 0 1 1 1 1 1 0 0 0 Y 1 1 BR\2 (V mode form)  
 DISPLACEMENT\16

Calculates an effective address, EA. Subtracts the 32-bit integer in the location specified by EA from the contents of L. Stores the results in L. If the result is greater than  $(2^{31})-1$ , an integer exception occurs and the instruction loads bit 1 of L with a 1 and bits 2 to 32 with  $(\text{result} - (2^{31}))$ .

If the result is less than  $-(2^{31})$ , an integer exception occurs and the instruction loads bit 1 of L with a 0 and bits 2 to 32 with the negative of  $(\text{result} + (2^{31}))$ .

If no overflow occurs, the instruction resets CBIT to 0. The instruction loads LINK with the borrow bit. The condition codes reflect the outcome of the operation. (See Appendix A.)

If an integer exception occurs and bit 8 of the keys contains a 0, the instruction sets CBIT to 1. If bit 8 contains a 1, the instruction sets CBIT to 1 and causes an integer exception fault. See Chapter 10 of the System Architecture Reference Guide for more information.

► SCB  
 Set CBIT to 1  
 1 1 0 0 0 0 0 1 1 0 0 0 0 0 0 0 (S, R, V mode form)

Sets the value of CBIT to 1. The value of LINK is indeterminate.  
 Leaves the values of the condition codes unchanged.

► SGL  
 Enter Single Precision Mode  
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 (S, R mode form)

Enters single precision mode by resetting bit 2 of the keys to 0. Subsequent LDA, STA, ADD, and SUB instructions manipulate 16-bit integers. Leaves the values of CBIT, LINK, and the condition codes unchanged.

► SGT  
 Skip on A Greater Than 0  
 1 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 (S, R, V mode form)

Skips the next sequential 16-bit halfword if the contents of A are greater than 0. Leaves the values of CBIT, LINK, and the condition codes unchanged.

► SKP n  
 Skip  
 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 (S, R, V mode form)

Skips the next sequential 16-bit halfword if the specified condition is met. Leaves the values of CBIT, LINK, and the condition codes unchanged.

This instruction allows you to test for several conditions. The table below shows the conditions available to test and information about the associated instruction.

Table 2-8  
SKP Conditions

Mnem	Opcode	Condition
NOP	101000	No operation.
SKP	100000	Unconditional skip.
SLT	101400	Skip on bit 1 of A equal to 1.
SGE	100400	Skip on bit 1 of A equal to 0.
SLN	101100	Skip on bit 16 of A equal to 1.
SLZ	100100	Skip on bit 16 of A equal to 0.
SNE	101040	Skip on A not equal to 0.
SEQ	100040	Skip on A equal to 0.
SS1*	101020	Skip on sense switch 1 set to 1.
SR1*	100020	Skip on sense switch 1 reset to 0.
SS2*	101010	Skip on sense switch 2 set to 1.
SR2*	100010	Skip on sense switch 2 reset to 0.
SS3*	101004	Skip on sense switch 3 set to 1.
SR3*	100004	Skip on sense switch 3 reset to 0.
SS4*	101002	Skip on sense switch 4 set to 1.
SR4*	100002	Skip on sense switch 4 reset to 0.
SSS*	101036	Skip on any sense switches set to 1.
SSR*	100036	Skip on all sense switches reset to 0.
SSC	101001	Skip on CBIT set to 1.
SRC	100001	Skip on CBIT reset to 0.

#### Note

\*These are restricted instructions.

You do not have to specify the unique mnemonic to test a particular condition; you can specify the SKP mnemonic and give the correct bit configuration for bits 7 to 16 of the desired test. Make sure that you set bit 7 of the SKP instruction properly: if it contains a 1, the skip occurs if any of the specified conditions are true; if it contains a 0, the skip occurs if all of the specified conditions are false.

► SKS function,device  
Skip on Condition Satisfied  
0 1 1 1 0 0 FUNCTION\4 DEVICE\6 (S, R mode form)

Tests for the condition specified in the function field of the instruction. Leaves the values of CBIT, LINK, and the condition codes unchanged. See Chapter 11 of the System Architecture Reference Guide for more information.

Note

SKS is a restricted instruction.

- **SLE**  
Skip if A Less Than or Equal to 0  
1 0 0 0 0 0 1 0 1 0 0 1 0 0 0 0 (S, R, V mode form)

Skips the next sequential 16-bit halfword if the contents of A are less than or equal to 0. Leaves the values of CBIT, LINK, and the condition codes unchanged.

- **SLN**  
Skip on LSB of A Nonzero  
1 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 (S, R, V mode form)

Skips the next sequential 16-bit halfword if bit 16 of A is 1. Leaves the values of CBIT, LINK, and the condition codes unchanged.

- **SLZ**  
Skip on LSB of A Zero  
1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 (S, R, V mode form)

Skips the next sequential 16-bit halfword if the bit 16 in A equals 0. Leaves the values of CBIT, LINK, and the condition codes unchanged.

- **SMCR**  
Skip on Machine Check Reset to 0  
1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 (S, R, V mode form)

Skips the next 16-bit halfword if the machine check flag is 0. Leaves the values of CBIT, LINK, and the condition codes unchanged.

Note

If the processor is operating in machine check mode, this instruction has no meaning; it executes as an unconditional skip.

► **SMCS**  
 Skip on Machine Check Set to 1  
 1 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 (S, R, V mode form)

Skips the next 16-bit halfword if the machine check flag is 1. Leaves the values of CBIT, LINK, and the condition codes unchanged.

#### Note

If the processor is operating in machine check mode, this instruction has no meaning; it executes as a NOP.

► **SMI**  
 Skip on A Minus  
 1 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 (S, R, V mode form)

Skips the next sequential 16-bit halfword if the contents of A are less than 0. Leaves the values of CBIT, LINK, and the condition codes unchanged.

► **SNZ**  
 Skip on A Nonzero  
 1 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 (S, R, V mode form)

Skips the next sequential 16-bit halfword if the contents of A are not equal to 0. Leaves the values of CBIT, LINK, and the condition codes unchanged.

► **SPL**  
 Skip on A Plus  
 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 (S, R, V mode form)

Skips the next sequential 16-bit halfword if the contents of A are greater than or equal to 0. Leaves the values of CBIT, LINK, and the condition codes unchanged.

► **SRC**  
 Skip on CBIT Reset to 0  
 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 (S, R, V mode form)

Skips the next sequential 16-bit halfword if the value of CBIT is 0. Leaves the values of CBIT, LINK, and the condition codes unchanged.

► SSC  
 Skip on CBIT Set to 1  
 1 0 0 0 0 0 1 0 0 0 0 0 0 0 1 (S, R, V mode form)

Skips the next sequential 16-bit halfword if the value of CBIT is 1.  
 Leaves the values of CBIT, LINK, and the condition codes unchanged.

► SSM  
 Set the Sign of A Minus  
 1 1 0 0 0 0 0 1 0 1 0 0 0 0 0 0 (S, R, V mode form)

Sets bit 1 of A to 1. Leaves the values of CBIT, LINK, and the condition codes unchanged.

► SSP  
 Set the Sign of A Plus  
 1 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 (S, R, V mode form)

Sets bit 1 of A to 0. Leaves the values of CBIT, LINK, and the condition codes unchanged.

► SSSN  
 Store System Serial Number  
 0 1 0 0 0 0 0 0 1 1 0 0 1 0 0 0 (V mode form)

This instruction is applicable only for the 2350 to the 9955 II. A 14-character system identifier programmed into the processor during manufacturing consists of a 2-character plant location code followed by a 12-digit number. (These characters and numbers are in 7-bit ASCII format.) SSSN writes this system identifier into a 16-halfword block at the address specified by the XB register. (A halfword is 16 bits.) The first 8 halfwords of this block hold the system serial number string as provided by manufacturing; the remaining halfwords are reserved for future expansion and are 0.

Leaves the values of CBIT, LINK, and the condition codes indeterminate.

#### Note

If SSSN is used for any earlier system listed in "About This Book", an unimplemented instruction (UII) fault occurs. (See Chapter 10 of the System Architecture Reference Guide.)

► STA address  
 Store A Into Memory  
 I X 0 1 0 0 1 1 0 0 0 Y 0 0 BR\2 (V mode long)  
 DISPLACEMENT\16

I X 0 1 0 0 1 1 0 0 0 0 0 0 CB\2 (R mode long)  
 [ DISPLACEMENT\16 ]

I X 0 1 0 0 DISPLACEMENT\10 (S mode; R, V mode short)

Calculates an effective address, EA. Stores the contents of the A register in the location specified by EA. Leaves the values of CBIT, LINK, and the condition codes unchanged.

► STAC address  
 Store A Conditionally  
 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 (V mode form)  
 AP\32

Compares the contents of B with the contents of the location referenced by the specified address pointer. If the two values are equal, the instruction stores the contents of A into that referenced location. If the two values are not equal, execution continues with the next instruction. Leaves the values of CBIT and LINK unchanged. Sets the condition codes to EQ if the store occurs and to NE if not.

The comparison and store will not be separated by execution of other instructions. This means that no instruction can alter the contents of the specified memory location between the compare and the store.

#### Note

This instruction is useful when two cooperating, sequential processes are manipulating shared data. It is interlocked against direct memory I/O; this means you can use it to interlock a process with a DMA, DMC, or DMQ channel, as well as to interlock a memory location that is possibly accessed by I/O.

► STC flr  
 Store Character  
 0 0 0 0 0 0 1 0 1 1 0 1 FLR 0 1 0 (V mode form)

If the contents of the specified FLR are nonzero, the instruction stores the contents of bits 9 to 16 of A into the character byte pointed to by the appropriate FAR. Updates the contents of the appropriate FAR so that they point to the next character. Decrements the contents of the specified FLR by 1. Sets the condition code NE.

If the contents of the specified FLR are 0, the STC instruction sets the condition code EQ and does not store a character.

The STC instruction leaves the values of LINK and CBIT unchanged.

#### Note

When the instruction specifies FLRO, FARO is used; FLR1, FAR1.

► **STEX**  
Stack Extend  
0 0 0 0 0 0 1 0 1 1 0 0 1 1 0 1 (V mode form)

Extends the length of the procedure stack.

A and B contain a 32-bit number specifying the halfword size of the extension. (A halfword is 16 bits.)

The firmware rounds up the number specified by A and B to an even number of halfwords. The instruction uses this value to allocate a block of memory to the procedure stack. The extension and the initial stack do not have to be contiguous, since there may not have been enough room left in the initial stack to contain a complete frame.

The instruction returns a segment number/offset number in A and B that specifies the starting address of the extension.

The extension is automatically deallocated when the current procedure completes execution. There is no limit on the number of extensions you can make.

A stack fault occurs if there is no room for the extension. The values of CBIT, LINK, and the condition codes are indeterminate. See Chapters 8 and 10 of the System Architecture Reference Guide for more information about this instruction, stacks, and stack faults.

► **STFA far,address**  
Store FAR  
0 0 0 0 0 0 1 0 1 1 0 1 FAR 0 0 0 (V mode form)  
AP\32

Stores the specified FAR contents as a hardware recognizable indirect pointer at the memory location referenced by the specified address pointer. If the bit number field of that FAR contains 0, the instruction stores the first 32 bits (2 halfwords) of the pointer and clears the pointer's extend bit to 0. If the bit number field of that FAR does not contain 0, the instruction saves all 48 bits (three halfwords) of the pointer and sets the pointer's extend bit to 1. Leaves the values of CBIT, LINK, and the condition codes indeterminate.

► **STL address**  
 Store Long  
 I X 0 1 0 0 1 1 0 0 0 Y 1 1 BR\2 (V mode form)  
 DISPLACEMENT\16

Calculates an effective address, EA. Stores the contents of L in the 32-bit location specified by EA. Leaves the values of CBIT, LINK, and the condition codes unchanged.

► **STLC address**  
 Store L Conditionally  
 0 0 0 0 0 0 1 0 1 0 0 0 0 1 0 0 (V mode form)  
 AP\32

Calculates an effective address, EA. Stores the contents of L into the 32-bit location specified by EA if and only if the contents of the specified location equal the contents of E. Leaves the values of CBIT and LINK unchanged. The condition codes reflect the result of the comparison. (See Appendix A.)

#### Note

This instruction is useful when two cooperating, sequential processes are manipulating shared data. It is interlocked against direct memory I/O; this means you can use it to interlock a process with a DMA, DMC, or DMQ channel, as well as to interlock a memory location that is possibly accessed by I/O.

► **STLR address**  
 Store L Into Addressed Register  
 I X 0 0 1 1 1 1 0 0 0 Y 0 1 BR\2 (V mode form)  
 DISPLACEMENT\16

Calculates a 32-bit (1-word) effective address, EA. Stores the contents of L into the register location specified by the offset portion of EA. Bit 2 and bit 12 of the offset portion of EA determine the actions of this instruction as follows.

<u>Bit 2</u>	<u>Bit 12</u>	<u>Action</u>
1*	----	Ignore bit 1 and bits 3 to 9. The offset portion of EA specified an absolute register number from 0 to '377.
0*	1	Bits 13 to 16 of the offset portion of EA specify one of the registers '20 to '37 in the current register set.
0	0	Bits 13 to 16 of the offset portion of EA specify one of the registers 0 to '17 in the current register set.

\*This is a restricted instruction.

STLR leaves the values of CBIT and LINK unchanged; the values of the condition codes are indeterminate. See Chapter 9 of the System Architecture Reference Guide for more information about register sets.

#### Note

Do not use the STLR instruction to write into the keys or modals. You can use LPSW or a mode control operation to change either of these registers. Under no circumstances should you try to change the value of the current register set bits contained in the modals.

In addition, do not change the contents of the procedure base register (PB) with this instruction. Use either LPSW or a control transfer. Loading any value other than 0 into PBL will change future effective address calculations for the currently running process.



#### STPM

Store Processor Model Number

0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 (V mode form)

Stores the CPU model number and microcode revision number in an 8-halfword field. (A halfword is 16 bits.) XB contains a pointer to the field. The format of the field is shown in Table 2-9.

Table 2-9  
STPM Memory Field Format

Halfword	Name	Description
1 to 2	Processor Model Number	Contains a code specifying the machine: 0L - 400/500, no Rev B microcode 1L - 400, Rev. B microcode 2L - Reserved 3L - 350 4L - 450/550 5L - 750 6L - 650 7L - 250 8L - 850 9L - 250-II 10L - 550-II 11L - 2250 15L - 9950 16L - 9650 17L - 2550 18L - 9955 19L - 9750 21L - 2350 22L - 2655 23L - 9655 25L - 2450 30L - 9955 II 31L - 2755 34L - 6350 42L - 9755
3 to 4	Microcode Revision	Offset 3: Bits 1 to 8 Reserved Bits 9 to 16 Manufacturing microcode revision number Offset 4: Bits 1 to 16 Engineering microcode revision number
5	Processor Line	Specifies options enabled for this machine: Bits 1 to 15 Reserved; must be 0 Bit 16 Marketing segment specification bit
6	Extended Microcode ID	To be implemented.
7 to 8	---	Reserved for future use.

This instruction leaves the values of CBIT, LINK, and the condition codes unchanged.

#### Note

STPM is a restricted instruction.

► **STTM**  
 Store Process Timer  
 0 0 0 0 0 0 0 1 0 1 0 0 1 0 0 0 (V mode form)

Valid for the 550-II, 850, I450, and 2350 to 9955 II.

The current process time is represented by the sum of the 32-bit elapsed time (stored in the PCB) and the 32-bit interval timer (contained in the CPU hardware). Bit 17 of the elapsed time is equivalent in weight to bit 1 of the interval time. This operation is equivalent to the following sequence of instructions.

```
LDLR    PB% + '25    /* Get PCB address.
ADL      = '10L      /* Offset of elapsed time.
STL      TEMP1        /* Elapsed time address -> Temp.
LDLR    PB% + '30    /* Read timer.
IAB                      /* Store low order
STA      XB% + 2      /* 16 bits.
IAB                      /* Adjust
PIDA                      /* weighting.
ADL      TEMP1,*      /* Add elapsed time.
STL      XB% + 0
```

Leaves the values of the CBIT, LINK, and condition codes indeterminate. This instruction is not implemented on the 2250.

► **STX address**  
 Store X  
 I 0 1 1 0 1 1 1 0 0 0 Y 0 0 BR\2 (V mode long)  
 DISPLACEMENT\16  
  
 I 0 1 1 0 1 1 1 0 0 0 0 0 0 CB\2 (R mode long)  
 [ DISPLACEMENT\16 ]  
  
 I 0 1 1 0 1 DISPLACEMENT\10 (S mode; R, V mode short)

Calculates an effective address, EA. Stores the contents of X at the location specified by EA. Leaves the values of CBIT, LINK, and the condition codes unchanged.

#### Note

STX cannot directly specify indexing, though an address in the indirection chain may do so in 16S mode. See Appendix B for more information.

► STY  
 Store Y  
 I 1 1 1 0 1 1 1 0 0 0 Y 1 0 BR\2 (V mode form)  
 DISPLACEMENT\16

Calculates an effective address, EA. Stores the contents of Y at the location specified by EA. Leaves the values of CBIT, LINK, and the condition codes unchanged.

#### Note

The STY instruction cannot do indexing. See Appendix B for more information.

► SUB address  
 Subtract  
 I X 0 1 1 1 1 1 0 0 0 Y 0 0 BR\2 (V mode long)  
 DISPLACEMENT\16

I X 0 1 1 1 1 1 0 0 0 0 0 0 CB\2 (R mode long)  
 [ DISPLACEMENT\16 ]

I X 0 1 1 1 DISPLACEMENT\10 (S mode; R, V mode short)

Calculates an effective address, EA. Fetches the 16-bit integer contained in the location specified by EA and subtracts them from the contents of A. Stores the results in A.

If the result is greater than or equal to  $2^{*}15$ , an integer exception occurs and the instruction sets CBIT to 1 and loads bit 1 of A with a 1 and bits 2 to 16 with (result minus ( $2^{*}15$ )).

If the result is less than  $-2^{*}15$ , an integer exception occurs and the instruction loads bit 1 of A with 0 and bits 2 to 16 with the negative of (result + ( $2^{*}15$ )).

If no overflow occurs, the instruction resets CBIT to 0. LINK contains the carry-out bit. The condition codes reflect the result of the operation. (See Appendix A.)

If an integer exception occurs and bit 8 of the keys contains 0, the instruction sets CBIT to 1. If bit 8 contains a 1, the instruction sets CBIT to 1 and causes an integer exception fault. See Chapter 10 of the System Architecture Reference Guide for more information.

► SVC  
 Supervisor Call  
 0 0 0 0 0 0 0 1 0 1 0 0 0 1 0 1 (S, R, V mode form)

Supervisor call. Generates a directed fault. Leaves the values of CBIT, LINK, and the condition codes unchanged.

This instruction allows you to make an operating system request that is addressing mode independent. By software convention, this instruction sends an operation code and pointers to the operating system to generate a fault. For more information, refer to Chapter 10 of the System Architecture Reference Guide.

► SZE  
 Skip on A Zero  
 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 (S, R, V mode form)

Skips the next sequential 16-bit halfword if the contents of A equal 0. Leaves the values of CBIT, LINK, and the condition codes unchanged.

► TAB  
 Transfer A to B  
 1 1 0 0 0 0 0 0 1 1 0 0 1 1 0 0 (V mode form)

Transfers the contents of A into B. Leaves the values of CBIT, LINK, and the condition codes unchanged.

► TAK  
 Transfer A to Keys  
 0 0 0 0 0 0 1 0 0 0 0 0 1 1 0 1 (V mode form)

Moves a copy of the contents of A into the keys. Loads CBIT, LINK, and the condition codes as a result of the operation. Resets bits 15 to 16 of the keys to 0.

#### Note

If the new contents of the keys specifies a new addressing mode, the new mode takes effect with the instruction immediately following TAK.

► TAX  
 Transfer A to X  
 1 1 0 0 0 0 0 1 0 1 0 0 0 1 0 0 (V mode form)

Loads X with a copy of the contents of A. Leaves the values of CBIT, LINK, and the condition codes unchanged.

► TAY  
 Transfer A to Y  
 1 1 0 0 0 0 0 1 0 1 0 0 0 1 0 1 (V mode form)

Loads Y with a copy of the contents of A. Leaves the values of CBIT, LINK, and the condition codes unchanged.

► TBA  
 Transfer B to A  
 1 1 0 0 0 0 0 1 1 0 0 0 0 1 0 0 (V mode form)

Transfers a copy of the contents of B to A. Leaves the values of CBIT, LINK, and the condition codes unchanged.

**TCA**

Two's Complement A

1 1 0 0 0 0 0 1 0 0 0 0 0 1 1 1 (S, R, V mode form)

Forms the two's complement of the contents of A and stores the result in A. If the number to be complemented is  $-2^{15}$ , an integer exception occurs and the instruction loads  $-2^{15}$  into A. If no integer exception occurs, the instruction resets CBIT to 0. LINK contains the carry-out bit. The condition codes reflect the result of the operation. (See Appendix A.)

If an integer exception occurs and bit 8 of the keys contains 0, the instruction sets CBIT to 1. If bit 8 contains a 1, the instruction sets CBIT to 1 and causes an integer exception fault. See Chapter 10 of the System Architecture Reference Guide for more information.

**TCL**

Two's Complement Long

1 1 0 0 0 0 1 0 1 0 0 0 1 0 0 0 (V mode form)

Forms the two's complement of the contents of L and stores the result in L. If the number to be complemented is  $-2^{31}$ , an integer exception occurs and the instruction loads  $-2^{31}$  into L. If no integer exception occurs, the instruction resets CBIT to 0. LINK contains the carry-out bit. The condition codes reflect the result of the operation. (See Appendix A.)

If an integer exception occurs and bit 8 of the keys contains 0, the instruction sets CBIT to 1. If bit 8 contains a 1, the instruction sets CBIT to 1 and causes an integer exception fault. See Chapter 10 of the System Architecture Reference Guide for more information.

**TFLl flr**

Transfer FLR to L

0 0 0 0 0 0 1 0 1 1 0 1 FLR 0 1 1 (V mode form)

Transfers the contents of the specified FLR into L as an unsigned, 32-bit integer. Clears bits 1 to 11 of L to 0. Leaves the values of CBIT, LINK, and the condition codes unchanged.

**TKA**

Transfer Keys to A

0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 1 (V mode form)

Moves a copy of the keys into A. Leaves the values of CBIT, LINK, and the condition codes unchanged.

► TLFL flr  
 Transfer L to FLR  
 0 0 0 0 0 0 1 0 1 1 0 1 FLR 0 0 1 (V mode form)

Transfers the 32-bit unsigned integer contained in L into the specified FLR. Clears bits 1 to 11 of L to 0 so that bits 1 to 6 of the specified FLR will be 0. Leaves the values of CBIT, LINK, and the condition codes unchanged.

#### Note

This instruction allows you to load the specified FLR with a value computed at execution time. The maximum allowable integer you can load is  $2^{20}$ . This number is 21 bits wide and equals the number of bits in a 64K segment.

► TSTQ address  
 Test Queue  
 1 1 0 0 0 0 1 1 1 1 1 0 1 1 1 1 (V mode form)  
 AP\32

The address pointer in this instruction is to the QCB of a queue. This instruction tests the referenced queue and sets A to equal the number of items in the queue. Sets the condition codes to EQ when the queue is empty. If the queue is not empty, sets the condition codes to NE. Leaves the values of CBIT and LINK unchanged.

► TXA  
 Transfer X to A  
 1 1 0 0 0 0 1 0 0 0 0 1 1 1 0 0 (V mode form)

Transfers a copy of the contents of X to A. Leaves the values of CBIT, LINK, and the condition codes unchanged.

► TYA  
 Transfer Y to A  
 1 1 0 0 0 0 1 0 0 1 0 1 0 1 0 0 (V mode form)

Transfers a copy of the contents of Y to A. Leaves the values of CBIT, LINK, and the condition codes unchanged.

► WAIT address  
 Wait  
 0 0 0 0 0 0 0 0 1 1 0 0 1 1 0 1 (V mode form)  
 AP\32

The address pointer in this instruction is to a 16-bit semaphore counter, C. The instruction increments C. If C is greater than 0, either the resource is not available, or the event has not occurred. The instruction removes the PCB from the ready list, suspending the process, and adds it to the wait list associated with the semaphore. It then makes the register set available, turns off the process timer, and goes to the dispatcher to find another process to run. The dispatcher enables interrupts.

If C is less than or equal to 0, the currently executing process continues.

If the instruction places the PCB on the wait list, no general registers are saved. This means that a process cannot depend on these registers to be intact after this instruction occurs. This instruction potentially clears the general, floating, and XB registers.

Leaves CBIT, LINK, and the condition codes unchanged.

For more information about semaphores, the dispatcher, PCBs, and wait lists, refer to Chapter 9 of the System Architecture Reference Guide.

#### Note

This is a restricted instruction.

► XAD  
 Decimal Add  
 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 (V mode form)

Performs a decimal arithmetic operation under control of FAR0, FAR1, and L.

FAR0 contains the address of field 1. FAR1 contains the address of field 2. L contains the control word; fields B and C of the control word specify the decimal operation to be performed, as shown in Table 2-10.

Table 2-10  
 XAD Decimal Operations

B	C	Operation	Destination
0	0	+F1+F2	F2
0	1	+F1-F2	F2
1	0	-F1+F2	F2
1	1	-F1-F2	F2

The scale differential field in the control word specifies the difference in the decimal point alignment between F1 and F2:

SD	Relation of F1 and F2
SD>0	F1 > F2
SD=0	F1 = F2
SD<0	F1 < F2

If the T bit contains a 1, the results will be forced positive. For more information about decimal arithmetic, refer to Chapter 6 of the System Architecture Reference Guide.

If the add operation results in an overflow, a decimal exception occurs. If no overflow occurs, the instruction sets CBIT to 0 to indicate success.

If a decimal exception occurs and bit 11 of the keys contains a 0, the instruction sets CBIT to 1. If bit 11 contains a 1, the instruction sets CBIT to 1 and causes a decimal exception fault. See Chapter 10 of the System Architecture Reference Guide for more information.

## INSTRUCTION SETS GUIDE

The registers used are GR0, GR1, GR3 (E), GR4, GR6, FAR0, FAR1, FLR0, and FLR1. At the end of the XAD instruction, the contents of these registers is indeterminate. The value of LINK is indeterminate. The condition codes reflect the state of F2 after the decimal operation. (See Appendix A.)



### XBTD

Binary to Decimal Conversion

0 0 0 0 0 0 1 0 0 1 1 0 0 1 0 1 (V mode form)

Converts a binary number to a decimal number. FAR0 contains the decimal field address. L contains the control word.

This instruction uses fields A, E, and H in the control word. H specifies the length of the binary number and its location:

<u>H</u>	<u>Length</u>	<u>Location</u>
0	16 bits	EH register
1	32 bits	E register
2	64 bits	DAC register

Converts the specified binary integer to a decimal integer and stores the result in the location specified by FAR0. Overflow results in a decimal exception. If no overflow occurs, the instruction resets CBIT to 0. Leaves the value of LINK indeterminate. The values of the condition codes are indeterminate.

The registers used are GR0, GR1, GR3 (E), GR4, GR6, FAR0, and FLR0. At the end of the instruction, the contents of these registers are indeterminate.

When the source register contains a null string, the destination register will contain all zeros.

If a decimal exception occurs and bit 11 of the keys contains a 0, the instruction sets CBIT to 1. If bit 11 contains a 1, the instruction sets CBIT to 1 and causes a decimal exception fault. See Chapter 10 of the System Architecture Reference Guide for more information.

### Note

This instruction does not use or modify FAR1, FLR1, or FAC1.

► XCA  
 Exchange and Clear A  
 1 1 0 0 0 0 0 0 0 1 0 0 0 1 0 0 (S, R, V mode form)

Interchanges the contents of registers A and B, then clears A to 0. Leaves the values of CBIT, LINK, and the condition codes unchanged.

► XCB  
 Exchange and Clear B  
 1 1 0 0 0 0 0 0 1 0 0 0 0 1 0 0 (S, R, V mode form)

Interchanges the values of A and B and then clears B to 0. Leaves the values of CBIT, LINK, and the condition codes unchanged.

► XCM  
 Decimal Compare  
 0 0 0 0 0 0 1 0 0 1 0 0 0 0 1 0 (V mode form)

Compares two decimal numbers and sets the condition codes depending on the result of the compare.

FAR0 contains the address of field 1 (F1). FAR1 contains the address of field 2 (F2). L contains the control word. This instruction uses fields A, B, C, E, F, G, and H of the control word.

Compares the two specified numbers. The instruction uses the G field of the control field to adjust the two numbers before the compare:

<u>G</u>	<u>Decision</u>
>0	Low-order digits of F1 only affect the initial borrow from the low-order digit of F2.
<0	Assume F1 is zero-extended with low zeros.

The registers used are GR0, GR1, GR3 (E), GR4, GR6, FLR0, and FLR1. At the end of this instruction, the contents of these registers are indeterminate. The CBIT is reset to 0 when there is no decimal exception. (This instruction cannot cause a decimal exception.) Leaves the value of LINK indeterminate. The condition codes reflect the result of the compare, as follows.

<u>CC</u>	<u>Test Result</u>
GT	F2 > F1
EQ	F2 = F1
LT	F2 < F1

**XDIB**

Decimal to Binary Conversion

0 0 0 0 0 0 1 0 0 1 1 0 0 1 1 0 (V mode form)

Converts a decimal string to a binary string.

FAR0 contains the address of the decimal string. L contains the control word; this instruction uses the A, E, and H fields. Field H specifies the length of the binary string and its location:

<u>H</u>	<u>Length</u>	<u>Destination Register</u>
00	16 bits	A register
01	32 bits	L register
10	64 bits	L/E

Converts the decimal string to a binary string of the specified type and stores it in the specified register. A conversion error causes a decimal exception. Leaves the value of LINK unchanged. The values of the condition codes are indeterminate.

The registers used are GR0, GR1, GR3 (E), GR4, GR6, FAR0, and FLR0. At the end of this instruction the contents of these registers are indeterminate.

If a decimal exception occurs and bit 11 of the keys contains a 0, the instruction sets CBIT to 1. If bit 11 contains a 1, the instruction sets CBIT to 1 and causes a decimal exception fault. See Chapter 10 of the System Architecture Reference Guide for more information.

Note

This instruction does not use or modify FAR1, FLR1, or FAC1.

► XDV  
 Decimal Divide  
 0 0 0 0 0 0 1 0 0 1 0 0 0 1 1 1 (V mode form)

Divides a decimal number, D2, by another, D1, and stores the quotient and remainder in the location of D2.

FAR0 contains the address of D1. FAR1 contains the address of D2. L contains the control word; this instruction uses fields A, B, C, E, F, H, and T.

Both decimal numbers must be in trailing sign embedded format. In addition, D2 must contain a number of leading zeros equal to the length of D1.

The instruction divides the two numbers. After the divide, the location of D2 contains the quotient of length (D2 length - D1 length) followed by the remainder of length (D1 length). Since D2 had leading zeros, no overflow can occur.

If the T bit contains a 1, the results will be forced positive. For more information about decimal arithmetic, refer to Chapter 6 of the System Architecture Reference Guide.

The registers used are GRO, GR1, GR3 (E), GR4, GR6, FAR0, FAR1, FLRO, and FLR1. At the end of this instruction, the contents of these registers are indeterminate.

If D1 is 0, overflow occurs which causes a decimal exception. Decimal exceptions also occur if D1 or D2 have the incorrect data type or if the length of D2 is less than that of D1. If no overflow occurs, CBIT is reset to 0. At the end of the instruction, LINK and the condition codes contain undefined results.

If a decimal exception occurs and bit 11 of the keys contains a 0, the instruction sets CBIT to 1. If bit 11 contains a 1, the instruction sets CBIT to 1 and causes a decimal exception fault. See Chapter 10 of the System Architecture Reference Guide for more information.

► XEC address  
 Execute  
 I X 0 0 0 1 1 1 0 0 0 Y 1 0 BR\2 (V mode long)  
 DISPLACEMENT\16  
  
 I X 0 0 0 1 1 1 0 0 0 0 1 0 CB\2 (R mode long)  
 [ DISPLACEMENT\16 ]

Calculates an effective address, EA. Executes the instruction found at EA, but does not transfer control to that location. Leaves the values of CBIT, LINK, and the condition codes modified as specified by the executed instruction.

The XEC instruction has limited application since all instructions cannot be executed in this way. The XEC instruction is useful for 16-bit register generic instructions such as shifts, rotates, clears, interchanges, and NOPs.

The following instruction types should not be used with XEC since they may not execute properly or will produce undefined results: instructions that change the address mode, program counter, or instruction stream; instructions that cause arithmetic faults; decimal or character instructions; and generic skips.

► XED  
 Numeric Edit  
 0 0 0 0 0 0 1 0 0 1 0 0 1 0 1 0 (V mode form)

Edits the contents of a string under control of a subprogram.

The registers used are L, XB, FARO, FAR1, and FLRO. At the end of the instruction, the contents of these registers and the CBIT, LINK, and condition codes are indeterminate.

FARO contains the address of the source string. The source string must be leading separate sign type and must have at least the same number of decimal digits and the decimal point alignment as called for in the edit subprogram.

FAR1 contains the address of the destination string. Bits 1 to 8 of A contain the floating character; bits 9 to 16, the status register. Bits 1 to 8 of B contain the number of remaining bytes to be processed (used if a fault or interrupt occurs). Bits 9 to 16 of B contain the suppression character whose initial value is determined by bit 12 of the keys ('240 if bit 1 contains 0; '40 if bit 12 contains 1). XB contains the address of the edit subprogram.

The instruction uses an edit subprogram to alter a source string and store the edit result in a destination location(s). To set up, perform a decimal move to correct the type, alignment, and length of the number to be edited. Next, use a LCEQ instruction to set up the initial contents of the register.

Each 16-bit halfword in the edit subprogram has the format shown in Figure 2-6.

1	2	3	4	8	9	16
L	00		E		M	

Edit Subprogram Halfword Format  
Figure 2-6

where L is 1 if this 16-bit halfword is the last halfword  
in the subprogram,  
0 if it is not the last halfword;  
E is a suboperator;  
M is a suboperator modifier.

The XED instruction uses several variables internally to control the edit subprogram. These are shown in Table 2-11.

Table 2-11  
XED Internal Variables

Var	Definition
SC	Zero suppression character; contained in B. Initial value is the space character ('240 or '40, depending on whether bit 12 of the keys contains 0 or 1.
FC	Floating edit character; contained in A. Initial value is not defined.
SIGN	Sign of the source field. The first character fetch sets up the value of this variable.
SIG	End zero suppression flag.

There are 17 edit suboperators, shown in Table 2-12.

Table 2-12  
XED Suboperators

Subop	Mnem	Name and Description
00	ZS	Zero Suppress. Fetches M digits from the source field consecutively, each time checking SIG. If SIG is 1, copies the digit into the destination string. If SIG is 0 and the digit is not 0, inserts the floating character (if defined) and copies the digit into the destination field. If SIG is 0, the digit is not 0, and the floating character is not defined, sets the SIG flag and copies the digit into the destination. If SIG and the digit are both 0, substitutes SC for the 0 digit in the destination field.
01	IL	Insert Literal. Copies M into the destination string. Increments XB and FAR1 by 1.
02	SS	Set Suppress Character. Sets SC to M and increments XB by 1.
03	ICS	Insert Character. If SIG is 1, copies M into the destination string. If SIG is 0, copies SC into the destination string. Increments XB and FAR1 by 1.
04	ID	Insert Digits. If SIG is 0, and FC is defined, copies FC and M digits into the destination field then sets SIG to 1. Increments XB by 1, FAR0 by M, and FAR1 by M+1. If SIG is 0 and FC is not defined, sets SIG to 1 and copies M digits from the source to the destination; increments XB by 1 and both FAR0 and FAR1 by M. If SIG is 1, copies M digits from the source to the destination and increments XB by 1 and both FAR0 and FAR1 by M.
05	ICM	Insert Character if Minus. If SIGN = 1, copies M into the destination string. If SIGN = 1, copies SC into the destination string. Increments both SB and FAR1 by 1.
06	ICP	Insert Character if Plus. If SIGN = 0, copies M into the destination string. If SIGN = 1, copies SC into the destination string. Increments both SB and FAR1 by 1.
07	SFC	Set Floating Character. Sets FC to M and increments XB by 1.
10	SFP	Set Floating if Plus. If SIGN = 0, sets FC to M. If SIGN = 1, sets FC to SC. Increments XB by 1.
11	SFM	Set Floating if Minus. If SIGN = 1, sets FC to M. If SIGN = 0, sets FC to SC. Increments XB by 1.

Table 2-12  
XED Suboperators (continued)

Subop	Mnem	Name and Description
12	SFS	Set Floating to SIGN. If SIGN = 0, sets FC to '253. If SIGN = 1, sets FC to '255. Increments XB by 1.
13	JZ	Jump if Zero. If the condition flag in A = 0, increments XB by 1. If the condition flag in A = 1, adds M to XB and then increments XB by 1.
14	FS	Fill with Suppression Characters. Copies SC M times into the destination string. Increments XB by 1 and FAR1 by M.
15	SF	Set Significance. If SIG = 0 and FC <> 0, inserts FC into the destination string, sets SIG to 1, and increments XB and FAR1 by 1. If SIG = 0 and FC = 0, sets SIG to 1 and increments XB and FAR1 by 1. If SIG = 1, increments XB by 1.
16	IS	Insert Sign. If SIGN = 0, copies '253 into the destination string. If SIGN = 1, copies '255 into the destination string. Increments XB by 1.
17	SD	Suppress Digits. Fetches M digits from the source string and checks if they are '260. If the source digit = '260, inserts SC into the destination string. If the source digit <> '260, copies the source digit into the destination string. Increments XB by 1 and both FAR0 and FAR1 by M.
20	EBS	Embed Sign. Fetches M digits from the source string. If SIGN = 0, copies each digit into the destination string. If SIGN = 1, embeds a minus sign into each digit before copying it into the destination string. Table 6-15 shows the characters used to represent the sign/digit combinations. A } symbol represents negative 0.

► XMP  
 Decimal Multiply  
 0 0 0 0 0 0 1 0 0 1 0 0 0 1 0 0 (V mode form)

Multiplies one decimal number, M, by another, D1, and stores the result in D2's location in memory. M is right justified in field D2 at the start of the operation.

FAR0 contains the address of D1. FAR1 contains the address of D2. L contains the control word; this instruction uses fields A, B, C, E, F, G, H, and T. Field G, the scale differential, must contain the number of decimal digits in M.

The number of decimal digits in D2 is greater than or equal to the number of decimal digits in D1 plus the number of decimal digits in M (specified by G). Normally, the digits to the left (more significant side) of M are zeros. If this is not the case, then a partial product field is added in.

The instruction multiplies M by D1 and stores the result in the location specified by FAR1. The result of the multiply is:

$D1 \times M + \text{partial product field}$

The partial product field is equal to:

$\text{length}(D2) - M.$

The partial product field is left justified in D2's location. The maximum partial product added in per traverse of the multiplicand is:

$\text{source digits} + \text{multiplier digits processed}$

There is also an implied weighting of the partial product field. The weighting is:

$10^{**} \text{multiplier digits}$

If the T bit is set to 1, the results are forced positive. See Chapter 6 of the System Architecture Reference Guide for more information about decimal arithmetic.

A decimal exception occurs if there are more potential or actual product digits than there is space in D2.

The registers used are GR0, GR1, GR3 (E), GR4, GR6, FAR0, FAR1, and XB. At the end of this instruction, the contents of these registers are indeterminate. Overflow causes a decimal exception; if no overflow occurs, resets CBIT to 0. LINK contains undefined results. At the end of the instruction, the condition codes reflect the state of the result. (See Appendix A.)

If a decimal exception occurs and bit 11 of the keys contains a 0, the XMP instruction sets CBIT to 1. If bit 11 contains a 1, the

instruction sets CBIT to 1 and causes a decimal exception fault. See Chapter 10 of the System Architecture Reference Guide for more information.

► XMV  
 Decimal Move  
 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 1 (V mode form)

Moves a string of characters from one location to another.

FAR0 contains the address of the source string. FAR1 contains the address of the destination string. L contains the control word; this instruction uses fields A, B, D, E, F, G, H and T.

The instruction moves the contents of the source field into the destination field from right to left. If the B field in the control word is 1, changes the the sign of the source field during the move. If the D field in the control word is 1 and the scale differential is greater than 0, the instruction rounds the source field during the move. If the scale differential (from the H field) is less than 0, the instruction pads the source field with SD trailing zeros before transferring.

Since the T bit is used by all systems for this instruction, the result is forced positive if this bit is set to 1.

The registers used are GR0, GR1, GR2 (L), GR3 (E), GR4, GR6, FAR0, FAR1, FLR0, and FLR1. At the end of this instruction, the contents of these registers are indeterminate.

A decimal exception occurs if there are more non-zero source digits than there is room in the destination, after any padding. If there is no decimal exception, CBIT is reset to 0. Leaves the value of LINK indeterminate. The values of the condition codes reflect the state of the destination field after the move. (See Appendix A.)

If a decimal exception occurs and bit 11 of the keys contains a 0, the instruction sets CBIT to 1. If bit 11 contains a 1, the instruction sets CBIT to 1 and causes a decimal exception fault. If no exception occurs, the instruction sets CBIT to 0. See Chapter 10 of the System Architecture Reference Guide for more information about decimal exceptions.

#### Note

The source and destination strings may not overlap in memory.

**ZCM**

Compare Character Field

0 0 0 0 0 0 1 0 0 1 0 0 1 1 1 1 (V mode form)

Compares two fields and sets the condition codes depending on the result of the compare.

FAR0 contains the address of field 1 (F1). FLR0 contains an integer specifying the length of F1. FAR1 contains the address of field 2 (F2). FLR1 contains an integer specifying the length of F2.

The instruction compares the contents of F1 and F2 on a byte by byte basis. If the fields are not of equal length, the instruction automatically extends the shorter string with space characters. A space character is '240 or '40 when bit 12 of the keys contains 0 or 1, respectively. Sets the condition codes as a result of the compare:

<u>Result of Compare</u>	<u>Set Condition Codes</u>
F1 > F2	GT
F1 = F2	EQ
F1 < F2	LT

The registers used are GR3 (E), GR4, FAR0, FAR1, FLR0, and FLR1; at the end of this instruction, the contents of these registers are indeterminate.

When the instruction completes execution, the values of CBIT and LINK are indeterminate.

Note

This instruction uses GR3, GR4, the FARs, and the FLRs during its operation. Since ZCM does not save the contents of these registers before using them, any data contained in them is overwritten when this instruction executes, unless you save it ahead of time.

**ZED**

Character Field Edit

0 0 0 0 0 0 1 0 0 1 0 0 1 0 0 1 (V mode form)

Controls an edit subprogram.

Uses the registers FAR0, FAR1, FLR0, and XB. At the end of this instruction the contents of these registers are indeterminate. Leaves the values of CBIT, LINK, and the condition codes indeterminate.

FARO contains the address of the source string. FLRO specifies the length of the source string. FAR1 contains the address of the destination string. XB contains the address of the edit subprogram.

The instruction uses the edit subprogram to alter the source string, then loads the edited result into the destination string. The subprogram, addressed by the contents of XB, contains a list of commands, each with the format shown in Figure 2-7:

1	2	6	7	8	9	16
L	00000	E		M		

ZED Subprogram Word Format  
Figure 2-7

where L is 1 if this command is the last command in the subprogram,  
           0 if it is not;  
 E is the edit opcode;  
 M is the edit modifier.

Bits 2 to 6 must be 0.

M, the operator modifier, specifies information E uses when editing the source string. (See Table 2-13.)

E, the edit suboperator, specifies the operation to be performed on the source string. Available values for E are shown in Table 2-13.

Table 3-16  
ZED Suboperators

Subop	Value	Action
CPC	00	Copies characters from the source string into the destination string. If the length of the source string is greater than the contents of the M field, then CPC moves a total of M source characters into the destination string, increments FAR0 and FAR1 by M, increments XB by 1, and decrements FLRO by M. If the length of the source string is less than the contents of the M field, then CPC moves the rest of the source string into the destination string, and then pads the remaining space to be filled with spaces. (See note.) Increments FAR0 by FLRO and FAR1 by M, increments XB by 1, and decrements FLRO by FLRO (so FLRO = 0).
INL	01	Inserts M into the destination string and increments both XB and FAR1 by 1.
SKC	10	Skips characters in the source string. If the remaining length of the source string is greater than or equal to the contents of the M field, then SKC skips over the next M characters of the source field by incrementing FAR0 by M and decrementing FLRO by M. If the remaining length of the source string is less than the contents of the M field, SKC skips over FLRO characters in the source string by incrementing FAR0 by FLRO and decrementing FLRO by FLRO (FLRO = 0). In either case, SKC increments XB by 1.
BLK	11	Inserts M spaces (see note) into the destination string, increments FAR1 by M, and increments XB by 1.

#### Note

A space is '240 or '40, depending on whether bit 12 of the keys is 0 or 1. This instruction uses GR3, GR4, the FARs, and the FLRs during its operation. Since ZED does not save the contents of these registers before using them, any data contained in them is overwritten when this instruction executes, unless you save it ahead of time.

► ZFIL  
 Fill Field With Character  
 0 0 0 0 0 0 1 0 0 1 0 0 1 1 1 0 (V mode form)

Stores a character into a series of destination bytes.

Bits 9 to 16 of L contain the character to be stored. FAR1 contains the starting address of the destination field (byte aligned). FLR1 contains an integer specifying the length of the destination field (in bytes).

The instruction stores the character specified in L in each byte of the destination field. If FLR1 contains 0, no operation takes place. Leaves the values of CBIT, LINK, and the condition codes indeterminate.

The registers used are GR3 (E), GR4, FAR0, FAR1, FLR0, and FLR1; at the end of this instruction, the contents of these registers are indeterminate.

#### Note

This instruction uses GR3, GR4, the FARs, and the FLRs during its operation. Since ZFIL does not save the contents of these registers before using them, any data contained in them is overwritten when this instruction executes, unless you save it ahead of time.

► ZMV  
 Move Character Field  
 0 0 0 0 0 0 1 0 0 1 0 0 1 1 0 0 (V mode form)

Moves a character field from one location to another.

FAR0 contains the address of the source string (byte aligned). FLR0 specifies the length in bytes, N, of the source string. FAR1 contains the address of the destination string (byte aligned). FLR1 specifies the length in bytes, M, of the destination string.

Compares N and M. If N is less than M, the instruction moves the contents of the source string into the destination string followed by M-N space characters. (A space character is '240 or '40 when bit 12 of the keys is 0 or 1, respectively.) If the destination string is shorter, the instruction moves the first M characters of the source string into the destination string.

When the instruction completes, the values of FAR0, FAR1, FLR0, FLR1, CBIT, LINK, and the condition codes are indeterminate.

Note

The ZMV instruction uses GR3, GR4, the FARs, and the FLRs during its operation. Since ZMV does not save the contents of these registers before using them, any data contained in them is overwritten when this instruction executes, unless you save it ahead of time. This instruction does not work with overlapping strings. See Chapter 6 of the System Architecture Reference Guide for more information.

**ZMVD**

Move Characters Between Equal Length Strings

0 0 0 0 0 0 1 0 0 1 0 0 1 1 0 1 (V mode form)

Moves characters from one string to another of equal length.

FAR0 contains the address of the source string. FAR1 contains the address of the destination string. FLR1 contains the number of characters to move, N.

The instruction moves N characters from the source string to the destination string. Characters are moved from lower addresses to higher addresses.

The registers used are GR3 (E), GR4, FAR0, FAR1, FLR0, and FLR1; at the end of this instruction, the contents of these registers are indeterminate. The values of CBIT, LINK, and the condition codes are indeterminate.

Note

The ZMV instruction uses GR3, GR4, the FARs, and the FLRs during its operation. Since ZMVD does not save the contents of these registers before using them, any data contained in them is overwritten when this instruction executes, unless you save it ahead of time. This instruction does not work with overlapping strings. See Chapter 6 of the System Architecture Reference Guide for more information.

► ZTRN  
 Character String Translate  
 0 0 0 0 0 0 1 0 0 1 0 0 1 0 0 0 (V mode form)

Translates a string of characters and stores the translations in the specified destination.

FAR0 contains the address of the source string (byte aligned). FAR1 contains the address of the destination string (byte aligned). FLR1 specifies the length of the source and destination strings. XB contains the starting address of a translation table. Each byte in the 256-byte table contains an alphabetic character.

The ZTRN instruction uses the address in FAR0 to reference a character. It interprets this character as an integer, adding it to the contents of XB to form an address into the translation table. The instruction takes the referenced character in the translation table and writes it into the location specified by FAR1. After storing the character, the instruction increments the contents of FAR0 and FAR1 by 1, decrements the contents of FLR1 by 1, and repeats the operation until FLR1 contains 0.

At the end of the instruction, FAR0 and FAR1 point to the location that follows the last byte of the source and destination strings, respectively. FLR1 contains 0. Leaves the values of XB, CBIT, LINK, and the condition codes unchanged.

#### Note

This instruction uses GR3, GR4, the FARs, and the FLRs during its operation. Since ZTRN does not save the contents of these registers before using them, any data contained in them is overwritten when this instruction executes, unless you save it ahead of time.

# 3

## I Mode

### INTRODUCTION

This chapter contains descriptions for all 50 Series instructions used in I mode. In the description of each instruction, you will find:

- The instruction mnemonic followed by any arguments.
- The name of the instruction.
- The bit format of the instruction.
- Detailed information describing the instruction's action.
- Information about the how the instruction affects LINK, CBIT, and the condition codes.

### Notation Conventions

Several abbreviations and symbols are used throughout this dictionary. Table 3-1 defines the dictionary notation.

Table 3-1  
Dictionary Notation

Symbol	Meaning
A	The 16-bit A register.
ADDRESS	Encompasses all the elements needed to specify an effective address. This term is used because various addressing types require you to specify the elements in different orders (such as indirect or pre- and post-indexing).
AP	Address pointer.
B	The 16-bit B register.
BR	Base register.
CBIT	Bit 1 of the keys.
DAC	The double precision floating-point accumulator with 48 bits of mantissa and 16 bits of exponent.
Displacement	The number of halfwords to be added to the base register to form the effective address.
DR	Destination register (normal register specifier).
E	The 32-bit E register.
EA	Effective address.
F	Floating-point accumulator.
FAC	The single precision floating-point accumulator with 48 bits of mantissa and 16 bits of exponent.
FAR	Field address register.
FLR	Field length register.
GRn	A 32-bit general register, where n is 0 through 7.
Halfword	A 16-bit unit of memory.
I	Indirect bit.
L	The 32-bit L register.
LINK	Bit 3 of the keys. Not used in S and R modes.

Table 3-1 (continued)  
Dictionary Notation

Symbol	Meaning
Offset	The number of halfwords from the starting address of a segment.
PB	The procedure base register.
QAC	The quad precision floating-point accumulator with 96 bits of mantissa and 16 bits of exponent.
R	A 32-bit general register.
r	Bits 1 to 16 of a general register.
skip	Skip next 16-bit halfword before continuing execution.
SR	Source register (or index if memory reference).
TM	Tag modifier. Bits used in I mode effective address calculation to specify indirection, indexing, etc.
X	The X register (indexing).
XB	Auxiliary base register.
Word	A 32-bit unit of memory.
Y	The Y register (indexing).
m\ <u>n</u>	Specifies the number of bits, <u>n</u> , occupied by field <u>m</u> .
[ ]	Specifies an optional argument.

### Resumable Instructions

Some assembly language instructions are resumable. When an interrupt is requested during the execution of an instruction, the processor usually services the interrupt at the end of execution before starting the next instruction. Some instructions, however, are too long or too complex for this to be desirable. When an interrupt is requested during one of these resumable instructions, the processor preserves the state of the interrupted instruction, handles the interrupt, then resumes the instruction at the point where the interrupt occurred. Table 3-2 lists the resumable assembly language instructions.

Table 3-2  
Resumable Instructions

Instructions			
ARGT	XAD	XBTD	XCM
XDTB	XDV	XED	XMP
XMV	ZCM	ZED	ZFIL
ZMV	ZMVD	ZTRN	STEX

These instructions depend on the settings in certain registers to determine whether they are being executed for the first or another time. In addition, some registers may be used for intermediate storage, modifying the previous contents as a side effect. Registers so modified are noted per instruction description.

#### Storing Data Into the 6350 and 9750 to 9955 II Instruction Stream

After any instruction that stores data into memory, you must wait five instructions before executing data. If in doubt about the next five instructions (temporally) to be executed, a mode change instruction to the current addressing mode, such as E32I, allows the stored data to be executed.

#### Instruction Formats

All I mode instructions belong to one of the following instruction types:

- I Mode Memory Reference
- I Mode Special Memory Reference
- I Mode Generic AP (Address Pointer)
- I Mode Register Generic
- I Mode Register Generic Branch
- Generic A and B (see below)

The format of each instruction type is shown in Figure 3-1.

Memory reference instructions have the opcode in bits 1 to 6. Special memory reference instructions (for floating point) have the opcode in bits 2, 3, 7, and 9; bit 8 specifies the floating accumulator. Some memory reference and special memory reference instructions have

register-to-register and/or immediate forms. Such instructions are so identified in this I Mode Instruction Dictionary.

The immediate form of a memory reference instruction has a 16-bit literal in bits 17 to 32 instead of a 16-bit displacement. Register-to-register forms are 16 bits long, since they have no displacement. Bits 7 to 9 specify the destination register and bits 12 to 14 specify the source register.

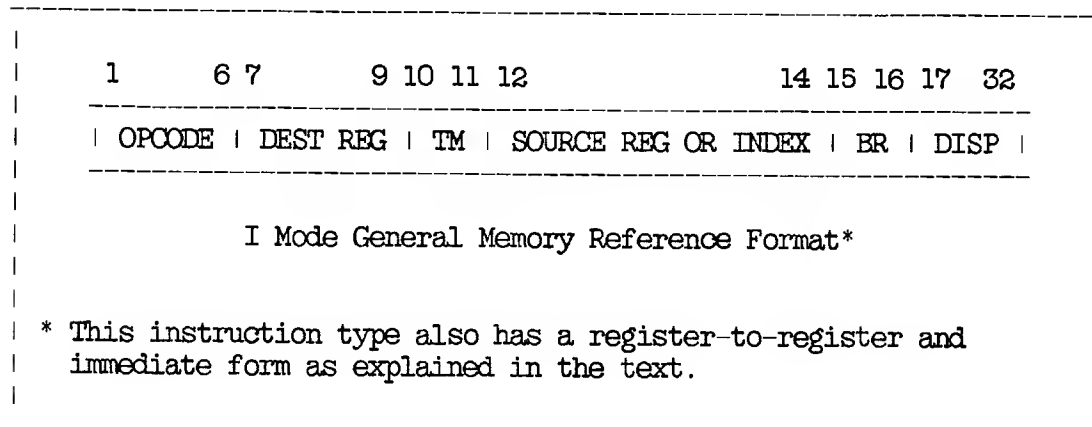
The immediate form of a special memory reference instruction has a 16-bit encoding in bits 17 to 32 instead of a 16-bit displacement. The register-to-register form is 16 bits long, since it has no displacement. Bit 8 specifies the floating-point destination accumulator and bits 12 to 14 specify the index register or the floating-point source register (in bit 13).

Generic AP instructions have a generic format (where bits 10 to 16 contain the opcode extension) followed by a 32-bit address pointer.

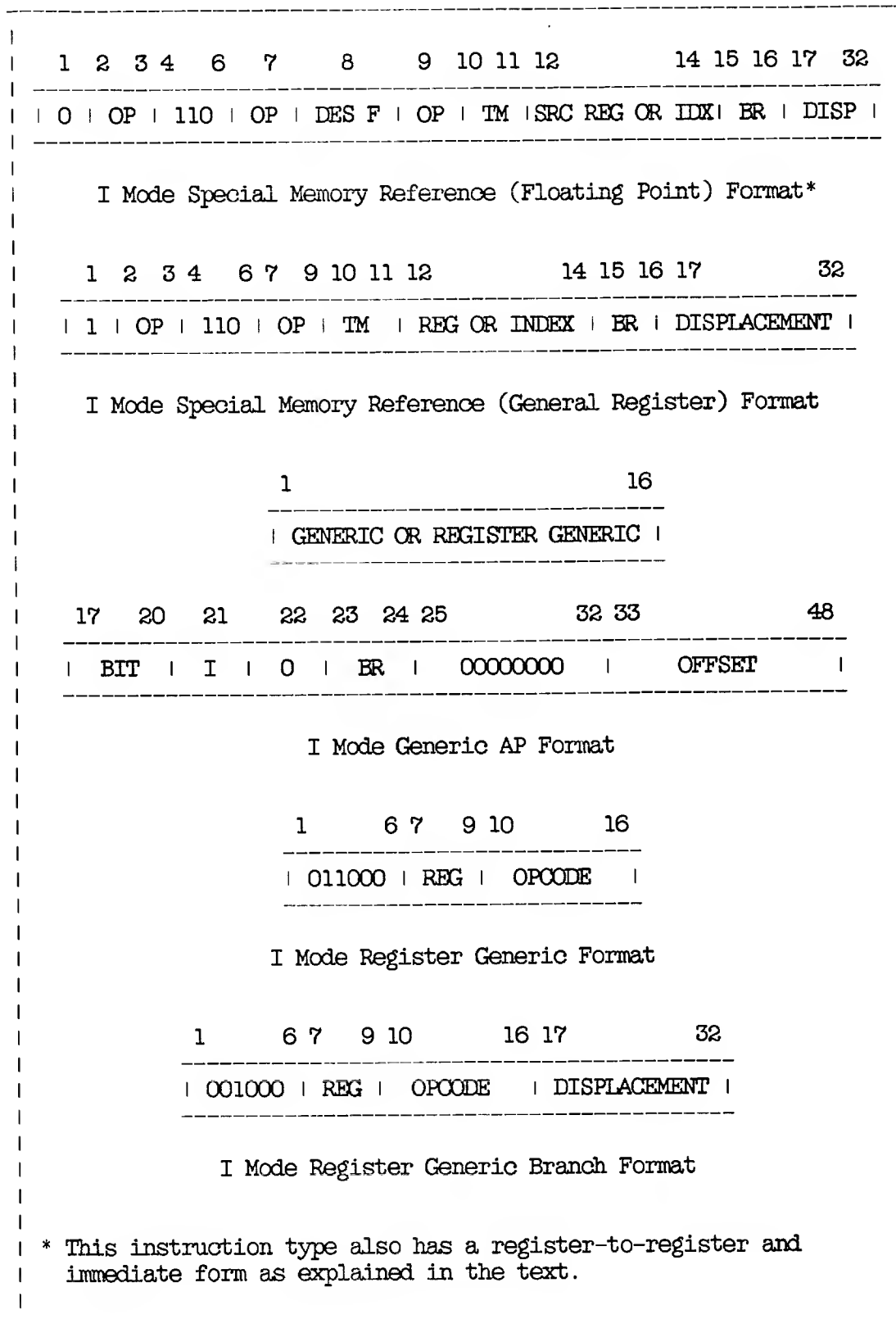
Register generic instructions are 16 bits long and have an opcode in bits 10 to 16. The value of bits 1 to 6 is 011000; bits 7 to 9 specify a general register.

Register generic branch instructions are 32 bits long and have an opcode in bits 10 to 16. The value of bits 1 to 6 is 00100; bits 7 to 9 specify a general register. Bits 17 to 32 contain a displacement.

Generic A and B instructions that do not reference the A, B, E, or L registers are also used in I Mode. See Chapter 2, Figure 2-1 for the format of these instructions. Instructions defined in I mode for this class are included in this instruction dictionary.



I Mode Instruction Formats  
Figure 3-1



I Mode Instruction Formats  
Figure 3-1 (continued)

INSTRUCTIONS

► A R, address  
 Add Fullword  
 0 0 0 0 1 0 DR\3 TM\2 SR\3 BR\2  
 [ DISPLACEMENT\16 ]

Calculates an effective address, EA. Fetches the 32-bit contents of the location specified by EA and adds them to the contents of the specified R. Stores the results in the specified R.

If the resulting sum is less than or equal to  $(2^{31})-1$  and greater than or equal to  $-(2^{31})$ , the instruction resets CBIT to 0. If the sum is greater than or equal to  $2^{31}$ , an integer exception occurs. If the sum is less than or equal to  $-(2^{31})-1$ , an integer exception occurs.

When an integer exception occurs, the results are of the opposite sign of the correct answer. In addition, the 32 bits are the 32 LSBs of the correct answer (that needs 33 bits to be correctly represented).

If an integer exception occurs and bit 8 of the keys contains a 0, the instruction sets CBIT to 1. If bit 8 contains a 1, the instruction sets CBIT to 1 and causes an integer exception fault. See Chapter 10 of the System Architecture Reference Guide for more information.

At the end of the operation, LINK contains the carry-out bit. The condition codes reflect the result of the operation. (See Appendix A.)

Note

This instruction also has a register-to-register and an immediate form. See Appendix B for more information.

► ABQ r, address  
 Add Entry to Bottom of Queue  
 0 1 1 0 0 0 R\3 1 0 1 1 1 0 0  
 AP\32

Adds the entry contained in the specified r to the bottom of the queue referenced by the AP. (AP points to the queue's QCB.) Sets the condition codes to reflect EQ if the queue was full, or to NE if not full. Leaves the values of CBIT and LINK unchanged.

► ACP destination-R, source-R  
 Add C Pointer  
 1 0 1 1 0 1 DR\3 TM\2 SR\3 BR\2

Adds the two's complement number contained in the specified source R to the C language pointer in the specified destination R. Stores the result in the C pointer in the same destination R. Leaves the values of the CBIT, LINK, and condition codes unchanged.

Addition is done to segment-number|offset|byte, producing a new pointer with an updated segment-number|offset|byte. Adding a positive integer that causes the segment-number field to overflow will modify the ring field. Adding a negative integer that causes the segment-number field to underflow will also modify the ring field. R contents that do not cause the segment number to overflow will not modify the ring field. No overflow is detected or indicated.

#### Note

While of the memory referencing form, this instruction is only defined for register-to-register and immediate address formation. (See Appendix B.)

If ACP is used for any earlier system listed in "About This Book", an unimplemented instruction (UII) fault occurs. (See Chapter 10 of the System Architecture Reference Guide.)

► ADLR R  
 Add LINK to Register  
 0 1 1 0 0 0 R\3 0 0 0 1 1 0 0

Adds the contents of LINK to the contents of R and stores the result in R. If there is an overflow, an integer exception occurs. If no integer exception occurs, CBIT is reset to 0. LINK contains the carry-out bit. The condition codes reflect the result of the operation. (See Appendix A.)

If an integer exception occurs and bit 8 of the keys contains 0, the instruction sets CBIT to 1. If bit 8 contains 1, the instruction sets CBIT to 1 and causes an integer exception fault. See Chapter 10 of the System Architecture Reference Guide for more information.

► AH r,address  
 Add Halfword  
 0 0 1 0 1 0 DR\3 TM\2 SR\3 BR\2  
 [ DISPLACEMENT\16 ]

Calculates an effective address, EA. Fetches the 16-bit contents of the location specified by EA and adds them to the contents of the specified r. Stores the results in the specified r.

If the resulting sum is less than or equal to  $(2^{15})-1$  and greater than or equal to  $-(2^{15})$ , the instruction resets CBIT to 0. If the sum is greater than or equal to  $2^{15}$ , an integer exception occurs. If the sum is less than or equal to  $-(2^{15})-1$ , an integer exception occurs.

When an integer exception occurs, the results are of the opposite sign of the correct answer. In addition, the 16 bits are the 16 LSBs of the correct answer (that needs 17 bits to be correctly represented).

If an integer exception occurs and bit 8 of the keys contains a 0, the instruction sets CBIT to 1. If bit 8 contains a 1, the instruction sets CBIT to 1 and causes an integer exception fault. See Chapter 10 of the System Architecture Reference Guide for more information.

At the end of the operation, LINK contains the carry-out bit. The condition codes reflect the result of the operation. (See Appendix A.)

#### Note

This instruction also has a register-to-register and an immediate form. See Appendix B for more information.

► AIP R,address  
 Add Indirect Pointer  
 1 1 1 1 0 1 DR\3 TM\2 SR\3 BR\2  
 [ DISPLACEMENT\16 ]

Adds the value contained in the specified R to the 32-bit value contained in the location specified by EA. Stores the result in the specified R. Checks these contents for a pointer fault.

This pointer fault is generated when the contents of the memory location to be added to the specified R contain a pointer fault (bit 1 contains 1).

If this pointer fault occurs, the pointer to the memory location is saved in FADDR (SB + 11) as well as bits 1 to 16 of the contents of that memory location FCODEH (SB + 10). After completion of the fault handling mechanism, the instruction can be re-executed. (See Chapter 10 of the System Architecture Reference Guide.)

An overflow produces an integer exception. If no integer exception occurs, CBIT is reset to 0. LINK contains the carry-out bit. The condition codes reflect the result of the operation. (See Appendix A.)

If an integer exception occurs and bit 8 of the keys contains 0, the instruction sets CBIT to 1. If bit 8 contains 1, the instruction sets CBIT to 1 and causes an integer exception fault. See Chapter 10 of the System Architecture Reference Guide for more information.

#### Note

AIP should weaken the ring field against the ring field of the effective address. This is not done on some current processors, but will be done on all future processors.

If AIP is used for any earlier system listed in "About This Book", an unimplemented instruction (UII) fault occurs. (See Chapter 10 of the System Architecture Reference Guide.)

► ARFA far,R  
Add Register to FAR  
0 1 1 0 0 0 R\3 1 1 1 FAR 0 0 1

Adds the bit address in the specified R to the contents of the specified FAR. Stores the result in the FAR. Leaves the values of CBIT and LINK indeterminate. Leaves the values of the condition codes unchanged.

► ARGV  
Argument Transfer  
0 0 0 0 0 0 0 1 1 0 0 0 0 1 0 1

Transfers arguments from a source procedure to a destination procedure. ARGV is fetched and executed only when the argument transfer phase of a procedure call (PCL) instruction is interrupted or faulted.

To perform a procedure call and argument transfer, the source procedure must contain the PCL instruction followed by a number of argument templates. The destination procedure must begin with the ARGV instruction. When the PCL instruction is executed, control transfers to the destination procedure, and the ARGV instruction uses the templates to form the actual arguments. The arguments are stored in the new stack frame as they are computed. At the end of the ARGV instruction, the values of CBIT, LINK, and the condition codes are indeterminate.

ARGV must be the first executable instruction in any destination procedure that will use arguments. For those procedures whose entry control blocks specify zero arguments, you must omit ARGV or you will

destroy the return pointer for PCL, producing indeterminate results. For information about argument transfers, refer to the procedure calls section in Chapter 8 of the System Architecture Reference Guide.

► ATQ r,address  
 Add Entry to Top of Queue  
 0 1 1 0 0 0 R\3 1 0 1 1 1 0 1  
 AP\32

Adds the entry contained in the specified r to the top of the queue referenced by the AP. (AP points to the queue's QCB.) Sets the condition codes to reflect EQ if the queue was full, or to NE if not full. Leaves the values of CBIT and LINK unchanged.

► BCEQ address  
 Branch on Condition Code EQ  
 1 1 0 0 0 0 1 1 1 0 0 0 0 0 1 0  
 ADDRESS\16

If the condition codes reflect equal to 0, the instruction loads the specified address into the program counter. This address must be within the current segment. If the condition codes reflect some other condition, execution continues with the next instruction. Leaves the values of CBIT, LINK, and the condition codes unchanged.

► BOGE address  
 Branch on Condition Code GE  
 1 1 0 0 0 0 1 1 1 0 0 0 0 1 0 1  
 ADDRESS\16

If the condition codes reflect greater than or equal to 0, the instruction loads the specified address into the program counter. This address must be within the current segment. If the condition codes reflect some other condition, execution continues with the next instruction. Leaves the values of CBIT, LINK, and the condition codes unchanged.

► BOGT address  
 Branch on Condition Code GT  
 1 1 0 0 0 0 1 1 1 0 0 0 0 0 0 1  
 ADDRESS\16

If the condition codes reflect greater than 0, the instruction loads the specified address into the program counter. This address must be within the current segment. If the condition codes reflect some other condition, execution continues with the next instruction. Leaves the values of CBIT, LINK, and the condition codes unchanged.

► BCLE address  
 Branch on Condition Code LE  
 1 1 0 0 0 0 1 1 1 0 0 0 0 0 0 0  
 ADDRESS\16

If the condition codes reflect less than or equal to 0, the instruction loads the specified address into the program counter. This address must be within the current segment. If the condition codes reflect some other condition, execution continues with the next instruction. Leaves the values of CBIT, LINK, and the condition codes unchanged.

► BCLT address  
 Branch on Condition Code LT  
 1 1 0 0 0 0 1 1 1 0 0 0 0 1 0 0  
 ADDRESS\16

If the condition codes reflect less than 0, the instruction loads the specified address into the program counter. This address must be within the current segment. If the condition codes reflect some other condition, execution continues with the next instruction. Leaves the values of CBIT, LINK, and the condition codes unchanged.

► BCNE address  
 Branch on Condition Code NE  
 1 1 0 0 0 0 1 1 1 0 0 0 0 0 1 1  
 ADDRESS\16

If the condition codes reflect not equal to 0, the instruction loads the specified address into the program counter. This address must be within the current segment. If the condition codes reflect some other condition, execution continues with the next instruction. Leaves the values of CBIT, LINK, and the condition codes unchanged.

► BCR address  
 Branch on CBIT Reset to 0  
 1 1 0 0 0 0 1 1 1 1 0 0 0 1 0 1  
 ADDRESS\16

If CBIT has the value 0, the instruction loads the specified address into the program counter. This address must be within the current segment. If CBIT has the value 1, execution continues with the next instruction. Leaves the values of CBIT, LINK, and the condition codes unchanged.

► BCS address  
 Branch on CBIT Set to 1  
 1 1 0 0 0 0 1 1 1 1 0 0 0 1 0 0  
 ADDRESS\16

If CBIT has the value 1, the instruction loads the specified address into the program counter. This address must be within the current segment. If CBIT has the value 0, execution continues with the next instruction. Leaves the values of CBIT, LINK, and the condition codes unchanged.

► BFEQ f,address  
 Branch on Floating Accumulator Equal to 0  
 0 0 1 0 0 0 0 F 0 1 0 1 0 0 1 0  
 ADDRESS\16

If the specified floating accumulator contents are equal to 0, BFEQ loads the specified address (in the current segment) into the program counter; if they are not equal to 0, execution continues with the next instruction. The condition codes reflect the comparison. (See Appendix A.) Leaves the LINK and CBIT unchanged. BFEQ works correctly only on normalized or nearly normalized numbers, because it checks the first 32 fraction bits only for equal to 0 and less than 0. (See the System Architecture Reference Guide, Chapter 6.)

► BFGE f,address  
 Branch on Floating Accumulator Greater Than or Equal to 0  
 0 0 1 0 0 0 0 F 0 1 0 1 0 1 0 1  
 ADDRESS\16

If the specified floating accumulator contents are greater than or equal to 0, BFGE loads the specified address (in the current segment) into the program counter; if they are less than 0, execution continues with the next instruction. The condition codes reflect the comparison. (See Appendix A.) Leaves the LINK and CBIT unchanged. BFGE works correctly only on normalized or nearly normalized numbers, because it checks the first 32 fraction bits only for equal to 0 and less than 0. (See the System Architecture Reference Guide, Chapter 6.)

► BFGT f,address  
 Branch on Floating Accumulator Greater Than 0  
 0 0 1 0 0 0 0 F 0 1 0 1 0 0 0 1  
 ADDRESS\16

If the specified floating accumulator contents are greater than 0, BFGT loads the specified address (in the current segment) into the program counter; if they are less than or equal to 0, execution continues with the next instruction. The condition codes reflect the comparison. (See Appendix A.) Leaves the LINK and CBIT unchanged. BFGT works correctly only on normalized or nearly normalized numbers, because it checks the first 32 fraction bits only for equal to 0 and less than 0. (See the System Architecture Reference Guide, Chapter 6.)

► BFLE f,address  
 Branch on Floating Accumulator Less Than or Equal to 0  
 0 0 1 0 0 0 0 F 0 1 0 1 0 0 0 0  
 ADDRESS\16

If the specified floating accumulator contents are less than or equal to 0, BFLE loads the specified address (in the current segment) into the program counter; if they are greater than 0, execution continues with the next instruction. The condition codes reflect the comparison. (See Appendix A.) Leaves the LINK and CBIT unchanged. BFLE works correctly only on normalized or nearly normalized numbers, because it checks the first 32 fraction bits only for equal to 0 and less than 0. (See Chapter 6 in the System Architecture Reference Guide.)

► BFLT f,address  
 Branch on Floating Accumulator Less Than 0  
 0 0 1 0 0 0 0 F 0 1 0 1 0 1 0 0  
 ADDRESS\16

If the specified floating accumulator contents are less than 0, BFLT loads the specified address (in the current segment) into the program counter; if they are greater than or equal to 0, execution continues with the next instruction. The condition codes reflect the comparison. (See Appendix A.) Leaves the LINK and CBIT unchanged. BFLT works correctly only on normalized or nearly normalized numbers, because it checks the first 32 fraction bits only for equal to 0 and less than 0. (See the System Architecture Reference Guide, Chapter 6.)

► BFNE f,address  
 Branch on Floating Accumulator Not Equal to 0  
 0 0 1 0 0 0 0 F 0 1 0 1 0 0 1 1  
 ADDRESS\16

If the specified floating accumulator contents are not equal to 0, BFNE loads the specified address (in the current segment) into the program counter; if they are equal to 0, execution continues with the next instruction. The condition codes reflect the comparison. (See Appendix A.) Leaves the LINK and CBIT unchanged. BFNE works correctly only on normalized or nearly normalized numbers, because it checks the first 32 fraction bits only for equal to 0 and less than 0. (See the System Architecture Reference Guide, Chapter 6.)

► BHD1 r,address  
 Branch on Half Register Decremented by 1  
 0 0 1 0 0 0 R\3 1 1 0 0 1 0 0  
 ADDRESS\16

Decrements the specified r contents by 1 and stores the result in the specified r. If the decremented value is not equal to 0, BHD1 loads the specified address (in the current segment) into the program counter. If that value is equal to 0, execution continues with the next instruction. Leaves the CBIT, LINK, and condition codes unchanged.

► BHD2 r,address  
 Branch on Half Register Decremented By 2  
 0 0 1 0 0 0 R\3 1 1 0 0 1 0 1  
 ADDRESS\16

Decrements the specified r contents by 2 and stores the result in the specified r. If the decremented value is not equal to 0, BHD2 loads the specified address (in the current segment) into the program counter. If that value is equal to 0, execution continues with the next instruction. Leaves the CBIT, LINK, and condition codes unchanged.

► BHD4 r,address  
 Branch on Half Register Decremented By 4  
 0 0 1 0 0 0 R\3 1 1 0 0 1 1 0  
 ADDRESS\16

Decrements the specified r contents by 4 and stores the result in the specified r. If the decremented value is not equal to 0, BHD4 loads the specified address (in the current segment) into the program counter. If that value is equal to 0, execution continues with the next instruction. Leaves the CBIT, LINK, and condition codes unchanged.

► BHEQ r,address  
 Branch on Half Register Equal To 0  
 0 0 1 0 0 0 R\3 1 0 0 1 0 1 0  
 ADDRESS\16

If the specified r contents are equal to 0, BHEQ loads the specified address (in the current segment) into the program counter; if they are not equal to 0, execution continues with the next instruction. Sets the condition codes to the comparison result. (See Appendix A.) Leaves the CBIT and LINK unchanged.

► **BHGE r,address**  
 Branch on Half Register Greater Than or Equal To 0  
 0 0 1 0 0 0 R\3 1 0 0 1 1 0 1  
 ADDRESS\16

If the specified r contents are greater than or equal to 0, BHGE loads the specified address (in the current segment) into the program counter; if they are less than 0, execution continues with the next instruction. Sets the condition codes to the result comparison. (See Appendix A.) Leaves the CBIT and LINK unchanged.

► **BHGT r,address**  
 Branch on Half Register Greater Than 0  
 0 0 1 0 0 0 R\3 1 0 0 1 0 0 1  
 ADDRESS\16

If the contents of the specified r are greater than 0, the instruction loads the specified address into the program counter. This address must be within the current segment. If the contents of r are less than or equal to 0, execution continues with the next instruction. Sets the condition codes to the result of the comparison. (See Appendix A.) Leaves the values of CBIT and LINK unchanged.

► **BHI1 r,address**  
 Branch on Half Register Incremented by 1  
 0 0 1 0 0 0 R\3 1 1 0 0 0 0 0  
 ADDRESS\16

Increments the contents of the specified r by 1 and stores the result in the specified r. If the incremented value is not equal to 0, the instruction loads the specified address into the program counter. This address must be within the current segment. If the incremented value is equal to 0, execution continues with the next instruction. Leaves the values of CBIT, LINK, and the condition codes unchanged.

► **BHI2 r,address**  
 Branch on Half Register Incremented by 2  
 0 0 1 0 0 0 R\3 1 1 0 0 0 0 1  
 ADDRESS\16

Increments the contents of the specified r by 2 and stores the result in the specified r. If the incremented value is not equal to 0, the instruction loads the the specified address into the program counter. This address must be within the current segment. If the incremented value is equal to 0, execution continues with the next instruction. Leaves the values of CBIT, LINK, and the condition codes unchanged.

► BHI4 r,address  
 Branch on Half Register Incremented by 4  
 0 0 1 0 0 0 R\3 1 1 0 0 0 1 0  
 ADDRESS\16

Increments the contents of the specified r by 4 and stores the result in the specified r. If the incremented value is not equal to 0, the instruction loads the specified address into the program counter. This address must be within the current segment. If the incremented value is equal to 0, execution continues with the next instruction. Leaves the values of CBIT, LINK, and the condition codes unchanged.

► BHLE r,address  
 Branch on Half Register Less Than or Equal to 0  
 0 0 1 0 0 0 R\3 1 0 0 1 0 0 0  
 ADDRESS\16

If the contents of the specified r are less than or equal to 0, the instruction loads the specified address into the program counter. This address must be within the current segment. If the contents of r are greater than 0, execution continues with the next instruction. Sets the condition codes to the result of the comparison. (See Appendix A.) Leaves the values of CBIT and LINK unchanged.

► BHLT r,address  
 Branch on Half Register Less Than 0  
 0 0 1 0 0 0 R\3 1 0 0 1 1 0 0  
 ADDRESS\16

If the contents of the specified r are less than 0, the instruction loads the specified address into the program counter. This address must be within the current segment. If the contents of r are greater than or equal to 0, execution continues with the next instruction. Sets the condition codes to the result of the comparison. (See Appendix A.) Leaves the values of CBIT and LINK unchanged.

► BHNE r,address  
 Branch on Half Register Not Equal To 0  
 0 0 1 0 0 0 R\3 1 0 0 1 0 1 1  
 ADDRESS\16

If the contents of the specified r are not equal to 0, the instruction loads the specified address into the program counter. This address must be within the current segment. If the contents of r are equal to 0, execution continues with the next instruction. Sets the condition codes to the result of the comparison. (See Appendix A.) Leaves the values of CBIT and LINK unchanged.

► BIR address  
 Branch on LINK Reset to 0  
 1 1 0 0 0 0 1 1 1 1 0 0 0 1 1 1  
 ADDRESS\16

If LINK has the value 0, the instruction loads the specified address into the program counter. This address must be within the current segment. If LINK has the value 1, execution continues with the next instruction. Leaves the values of CBIT, LINK, and the condition codes unchanged.

► BLS address  
 Branch on LINK Set to 1  
 1 1 0 0 0 0 1 1 1 1 0 0 0 1 1 0  
 ADDRESS\16

If LINK has the value 1, the instruction loads the specified address into the program counter. This address must be within the current segment. If LINK has the value 0, execution continues with the next instruction. Leaves the values of CBIT, LINK, and the condition codes unchanged.

► BMEQ address  
 Branch on Magnitude Condition EQ  
 1 1 0 0 0 0 1 1 1 0 0 0 0 0 1 0  
 ADDRESS\16

If the condition codes indicate magnitude equal to 0, the instruction loads the specified address into the program counter, like BCEQ. BMEQ is intended for magnitude comparisons after a compare or subtract instruction. This address must be within the current segment. If the condition codes indicate some other condition, execution continues with the next instruction. Leaves the values of CBIT, LINK, and the condition codes unchanged.

► BMGE address  
 Branch on Magnitude Condition GE  
 1 1 0 0 0 0 1 1 1 1 0 0 0 1 1 0  
 ADDRESS\16

If LINK has the value 1, the instruction loads the specified address into the program counter, like BLS. BMGE is used to determine if the magnitude of the register quantity was greater than or equal to the memory quantity after a compare or subtract instruction. This address must be within the current segment. If LINK has the value 0, execution continues with the next instruction. Leaves the values of CBIT, LINK, and the condition codes unchanged.

► **BMGT address**  
 Branch on Magnitude Condition GT  
 1 1 0 0 0 0 1 1 1 0 0 1 0 0 0  
 ADDRESS\16

If LINK is 1 and the condition codes reflect not equal to 0, the instruction loads the specified address into the program counter. This address must be within the current segment. If some other condition exists, execution continues with the next instruction. Leaves the values of CBIT, LINK, and the condition codes unchanged.

► **BMLE address**  
 Branch on Magnitude Condition LE  
 1 1 0 0 0 0 1 1 1 0 0 1 0 0 1  
 ADDRESS\16

If LINK is 0 or the condition codes reflect equal to 0, the instruction loads the specified address into the program counter. This address must be within the current segment. If some other condition exists, execution continues with the next instruction. Leaves the values of CBIT, LINK, and the condition codes unchanged.

► **BMLT address**  
 Branch on Magnitude Condition LT  
 1 1 0 0 0 0 1 1 1 0 0 0 1 1 1  
 ADDRESS\16

If LINK has the value 0, the instruction loads the specified address into the program counter, like BLR. BMLT is used to determine if the magnitude of the register quantity is less than the memory quantity after a compare or subtract instruction. This address must be within the current segment. If LINK has the value 1, execution continues with the next instruction. Leaves the values of CBIT, LINK, and the condition codes unchanged.

► **BMNE address**  
 Branch on Magnitude Condition NE  
 1 1 0 0 0 0 1 1 1 0 0 0 0 0 1 1 (V mode form)  
 ADDRESS\16

If the condition codes indicate magnitude not equal to 0, the instruction loads the specified address into the program counter, like BCNE. BMNE is intended for magnitude comparisons after a compare or subtract instruction. This address must be within the current segment. If the condition codes reflect some other condition, execution continues with the next instruction. Leaves the values of CBIT, LINK, and the condition codes unchanged.

► **BRBR R,bit #,address**  
 Branch on Register Bit Reset  
 0 0 1 0 0 0 R\3 0 1 BIT\5  
 ADDRESS\16

Bits 12 to 16 of the instruction contain a value between '00 and '37. This value specifies the bit position in the register to be tested. A value of '00 corresponds to bit 1; '01, bit 2; and so on.

If the specified bit position contains 0, the instruction loads the specified address into the program counter. This address must be within the current segment. If the specified bit position contains 1, execution continues with the next instruction. Leaves the values of CBIT, LINK, and the condition codes unchanged.

► **BRBS R,bit #,address**  
 Branch on Register Bit Set  
 0 0 1 0 0 0 R\3 0 0 BIT\5  
 ADDRESS\16

Bits 12 to 16 of the instruction contain a value between '00 and '37. This value specifies the bit position in the register to be tested. A value of '00 corresponds to bit 1; '01, bit 2; and so on.

If the specified bit position contains 1, the instruction loads the specified address into the program counter. This address must be within the current segment. If the specified bit position contains 0, execution continues with the next instruction. Leaves the values of CBIT, LINK, and the condition codes unchanged.

► **BRD1 R,address**  
 Branch on Register Decrement by 1  
 0 0 1 0 0 0 R\3 1 0 1 1 1 0 0  
 ADDRESS\16

Decrements the contents of the specified R by 1 and stores the result in the specified R. If the decremented value is not equal to 0, the instruction loads the specified address into the program counter. This address must be within the current segment. If the decremented value is equal to 0, execution continues with the next instruction. Leaves the values of CBIT, LINK, and the condition codes unchanged.

► BRD2 R,address  
 Branch on Register Decrement by 2  
 0 0 1 0 0 0 R\3 1 0 1 1 1 0 1  
 ADDRESS\16

Decrements the contents of the specified R by 2 and stores the result in the specified R. If the decremented value is not equal to 0, the instruction loads the specified address into the program counter. This address must be within the current segment. If the decremented value is equal to 0, execution continues with the next instruction. Leaves the values of CBIT, LINK, and the condition codes unchanged.

► BRD4 R,address  
 Branch on Register Decrement by 4  
 0 0 1 0 0 0 R\3 1 0 1 1 1 1 0  
 ADDRESS\16

Decrements the contents of the specified R by 4 and stores the result in the specified R. If the decremented value is not equal to 0, the instruction loads the specified address into the program counter. This address must be within the current segment. If the decremented value is equal to 0, execution continues with the next instruction. Leaves the values of CBIT, LINK, and the condition codes unchanged.

► BREQ R,address  
 Branch on Register Equal to 0  
 0 0 1 0 0 0 R\3 1 0 0 0 0 1 0  
 ADDRESS\16

If the contents of the specified R are equal to 0, the instruction loads the specified address into the program counter. This address must be within the current segment. If the R contents are not equal to 0, execution continues with the next instruction. Sets the condition codes to the result of the comparison. (See Appendix A.) Leaves the values of CBIT and LINK unchanged.

► ERGE R,address  
 Branch on Register Greater Than or Equal to 0  
 0 0 1 0 0 0 R\3 1 0 0 0 1 0 1  
 ADDRESS\16

If the contents of the specified R are greater than or equal to 0, the instruction loads the specified address into the program counter. This address must be within the current segment. If the R contents are less than 0, execution continues with the next instruction. Sets the condition codes to the result of the comparison. (See Appendix A.) Leaves the values of CBIT and LINK unchanged.

► **BRGT R,address**  
 Branch on Register Greater Than 0  
 0 0 1 0 0 0 R\3 1 0 0 0 0 1  
 ADDRESS\16

If the contents of the specified R are greater than 0, the instruction loads the specified address into the program counter. This address must be within the current segment. If the R contents are less than or equal to 0, execution continues with the next instruction. Sets the condition codes to the result of the comparison. (See Appendix A.) Leaves the values of CBIT and LINK unchanged.

► **BR11 R,address**  
 Branch on Register Incremented by 1  
 0 0 1 0 0 0 R\3 1 0 1 1 0 0 0  
 ADDRESS\16

Increments the contents of the specified R by 1 and stores the result in the specified R. If the incremented value is not equal to 0, the instruction loads the specified address into the program counter. This address must be within the current segment. If the incremented value is equal to 0, execution continues with the next instruction. Leaves the values of CBIT, LINK, and the condition codes unchanged.

► **BR12 R,address**  
 Branch on Register Incremented by 2  
 0 0 1 0 0 0 R\3 1 0 1 1 0 0 1  
 ADDRESS\16

Increments the contents of the specified R by 2 and stores the result in the specified R. If the incremented value is not equal to 0, the instruction loads the specified address into the program counter. This address must be within the current segment. If the incremented value is equal to 0, execution continues with the next instruction. Leaves the values of CBIT, LINK, and the condition codes unchanged.

► **BR14 R,address**  
 Branch on Register Incremented by 4  
 0 0 1 0 0 0 R\3 1 0 1 1 0 1 0  
 ADDRESS\16

Increments the contents of the specified R by 4 and stores the result in the specified R. If the incremented value is not equal to 0, the instruction loads the specified address into the program counter. This address must be within the current segment. If the incremented value is equal to 0, execution continues with the next instruction. Leaves the values of CBIT, LINK, and the condition codes unchanged.

► **BRLE R,address**  
 Branch on Register Less Than or Equal to 0  
 0 0 1 0 0 0 R\3 1 0 0 0 0 0 0  
 ADDRESS\16

If the contents of the specified R are less than or equal to 0, the instruction loads the specified address into the program counter. This address must be within the current segment. If the R contents are greater than 0, execution continues with the next instruction. Sets the condition codes to the result of the comparison. (See Appendix A.) Leaves the values of CBIT and LINK unchanged.

► **BRLT R,address**  
 Branch on Register Less Than 0  
 0 0 1 0 0 0 R\3 1 0 0 0 1 0 0  
 ADDRESS\16

If the contents of the specified R are less than 0, the instruction loads the specified address into the program counter. This address must be within the current segment. If the R contents are greater than or equal to 0, execution continues with the next instruction. Sets the condition codes to the result of the comparison. (See Appendix A.) Leaves the values of CBIT and LINK unchanged.

► **BRNE R,address**  
 Branch on Register Not Equal to 0  
 0 0 1 0 0 0 R\3 1 0 0 0 0 1 1  
 ADDRESS\16

If the contents of the specified R are not equal to 0, the instruction loads the specified address into the program counter. This address must be within the current segment. If the R contents are equal to 0, execution continues with the next instruction. Sets the condition codes to the result of the comparison. (See Appendix A.) Leaves the values of CBIT and LINK unchanged.

► C R, address  
 Compare Fullword  
 1 1 0 0 0 1 DR\3 TM\2 SR\3 BR\2  
 [ DISPLACEMENT\16 ]

Calculates an effective address, EA. Compares the 32-bit value contained in the specified R to the 32-bit value contained in the location specified by EA. The comparison is done by subtracting the contents of the the memory location from the contents of the register. Sets the condition codes to the result of the comparison. (See Appendix A.) Leaves the value of CBIT unchanged. LINK contains the carry-out bit.

#### Note

This instruction also has a register-to-register and an immediate form. See Appendix B for more information.

► CALF address  
 Call Fault Handler  
 0 0 0 0 0 0 0 1 1 1 0 0 0 1 0 1  
 AP\32

The address pointer in this instruction points to the ECB of a fault routine. CALF uses this pointer to transfer control to the fault routine as if the transfer were a normal procedure call with no arguments passed. The values of CBIT, LINK, and the condition codes are indeterminate. See Chapter 10 of the System Architecture Reference Guide for more information.

► CCP destination-R, source-R  
 Compare C Pointer  
 1 0 0 1 0 1 DR\3 TM\2 SR\3 BR\2

Compares the C language pointer in the specified source R to the C language pointer in the specified destination R. Ignores the pointer fault and ring bits during the comparison. Leaves the values of CBIT and LINK unchanged. Sets the condition codes to the outcome of the comparison as follows.

<u>Condition</u>	<u>CC</u>
Contents of destination-R > contents of source-R.	GT
Contents of destination-R = contents of source-R.	EQ
Contents of destination-R < contents of source-R.	LT

Note

While of the memory referencing form, the OCP instruction is only defined for register-to-register address formation. (See Appendix B of the Instruction Sets Guide.) The immediate form of this instruction is undefined, but the preferred implementation is a UII for the immediate form.

If OCP is used for any earlier system listed in "About This Book", an unimplemented instruction (UII) fault occurs. (See Chapter 10 of the System Architecture Reference Guide.)

► OGT r  
 Computed GOTO  
 0 1 1 0 0 0 R\3 0 0 1 0 1 1 0  
 INTEGER N\16  
 BRANCH ADDRESS 1\16  
 ...  
 BRANCH ADDRESS N-1\16

If the contents of the specified r are greater than or equal to 1 and less than the specified integer N that follows the opcode, the instruction adds the contents of r to the contents of the program counter to form an address. (The program counter points to the integer N following the opcode.) Loads the contents of the location specified by this address into the program counter. If the contents of r are not within this range, the instruction adds integer N to the contents of the program counter and stores the result in the program counter. Each of the branch addresses following the instruction specifies a location within the current procedure segment. The values of CBIT, LINK, and the condition codes are indeterminate.

► CH r, address  
 Compare Halfword  
 1 1 1 0 0 1 DR\3 TM\2 SR\2 BR\2  
 [ DISPLACEMENT\16 ]

Calculates an effective address, EA. Compares the value contained in the specified r to the 16-bit value contained in the location specified by EA. Leaves the value of CBIT unchanged. LINK contains the carry-out bit. The condition codes reflect the result of the comparison. (See Appendix A.)

Note

This instruction also has a register-to-register and an immediate form. See Appendix B for more information.

► CHS R  
Change Sign  
0 1 1 0 0 0 R\3 0 1 0 0 0 0 0

Complements bit 1 of the specified R. Leaves the values of CBIT, LINK, and the condition codes unchanged.

► CMH r  
Complement r  
0 1 1 0 0 0 R\3 0 1 0 0 1 0 1

Forms the one's complement of the contents of the specified r by inverting the value of each bit and stores the result in r. Leaves the values of CBIT, LINK, and the condition codes unchanged.

► CMR R  
Complement R  
0 1 1 0 0 0 R\3 0 1 0 0 1 0 0

Forms the one's complement of the contents of the specified R by inverting the value of each bit and stores the result in R. Leaves the values of CBIT, LINK, and the condition codes unchanged.

► CR R  
Clear R to 0  
0 1 1 0 0 0 R\3 0 1 0 1 1 1 0

Clears the specified R to 0. Leaves the values of CBIT, LINK, and the condition codes unchanged.

► CRBL R  
Clear R High Byte 1 Left  
0 1 1 0 0 0 R\3 0 1 1 0 0 1 0

Loads zeros into bits 1 to 8 of the specified R. Leaves the values of LINK, CBIT, and the condition codes unchanged.

► CRBR R  
Clear R High Byte 2 Right  
0 1 1 0 0 0 R\3 0 1 1 0 0 1 1

Loads zeros into bits 9 to 16 of the specified R. Leaves the values of LINK, CBIT, and the condition codes unchanged.

► CRHL R  
 Clear R Left Halfword  
 0 1 1 0 0 0 R\3 0 1 0 1 1 0 0

Clears bits 1 to 16 of the specified R to 0. Leaves the values of CBIT, LINK, and the condition codes unchanged.

► CRHR R  
 Clear R Right Halfword  
 0 1 1 0 0 0 R\3 0 1 0 1 1 0 1

Clears bits 17 to 32 of the specified R to 0. Leaves the values of CBIT, LINK, and the condition codes unchanged.

► CSR R  
 Copy Sign  
 0 1 1 0 0 0 R\3 0 1 0 0 0 0 1

Copies the value of bit 1 of the specified R into CBIT, and then loads 0 into bit 1 of R. The value of LINK is indeterminate. Leaves the condition codes unchanged.

► D R, address  
 Divide Fullword  
 1 1 0 0 1 0 DR\3 TM\2 SR\3 BR\2  
 [ DISPLACEMENT\16 ]

Calculates an effective address, EA. Divides the 64-bit value contained in the specified R and R+1 by the 32-bit value contained in the location specified by EA. Stores the quotient in the specified R and the remainder in R+1. Overflow may occur if the quotient is less than  $-(2^{31})$  or greater than  $(2^{31})-1$ . Overflow and divide by 0 cause an integer exception.

If no integer exception occurs, CBIT is reset to 0. The instruction leaves the values of LINK and the condition codes indeterminate.

If an integer exception occurs and bit 8 in the keys contains 0, the instruction sets CBIT to 1; if bit 8 contains 1, the instruction sets CBIT to 1 and causes an integer exception fault. For more information, see Chapter 10 of the System Architecture Reference Guide.

#### Note

R must specify an even register. This instruction also has a register-to-register and an immediate form. See Appendix B for more information.

► DBLE f  
 Convert Single to Double Floating Point  
 0 1 1 0 0 0 0 F 0 1 0 0 0 1 1 0

Converts the single precision number in the specified floating-point accumulator to a double precision one by zeroing bits 32 to 48 of the floating-point accumulator. Stores the result in the floating-point accumulator. Leaves the values of CBIT, LINK, and the condition codes unchanged. Overflow or underflow cannot occur.

► DCP R  
 Decrement C Pointer  
 0 1 1 0 0 0 R\3 1 1 1 0 0 0 0

Decrements the C language pointer in the specified R by 1 byte. Decrementing a 0 offset reduces the segment number by 1. Decrementing segment number 0, offset 0, byte 0 generates a pointer to the maximum segment number, the maximum offset, and byte 0. Leaves the CBIT, LINK, and the condition codes unchanged. For C pointer details, see Chapter 1 and Appendix B of this guide.

Note

If DCP is used for any earlier system listed in "About This Book", an unimplemented instruction (UII) fault occurs. (See Chapter 10 of the System Architecture Reference Guide.)

► DFA f,address  
Double Floating Add  
0 0 1 1 1 0 1 F 1 TM\2 SR\3 BR\2  
[ DISPLACEMENT\16 ]

Calculates an effective address, EA. Adds the contents of the specified DAC to the contents of the location specified by EA. Stores the result in the DAC. An overflow causes a floating-point exception. If no floating-point exception occurs, CBIT is reset to 0. The values of LINK and the condition codes are indeterminate.

If a floating-point exception occurs and bit 7 of the keys contains a 1, the instruction sets CBIT to 1. If bit 7 contains a 0, the instruction sets CBIT to 1 and causes a floating-point exception fault. See Chapter 10 of the System Architecture Reference Guide.

Note

This instruction also has a register-to-register and an immediate form. See Appendix B for more information.

► DFC f,address  
Double Floating Compare  
0 0 0 1 1 0 1 F 1 TM\2 SR\3 BR\2  
[ DISPLACEMENT\16 ]

Calculates an effective address, EA. Compares the contents of the specified DAC to the contents of the location specified by EA. Leaves the values of CBIT and LINK unchanged. Sets the condition codes to the outcome of the comparison.

<u>Condition</u>	<u>CC</u>
Contents of DAC > contents of location specified by EA.	GT
Contents of DAC = contents of location specified by EA.	EQ
Contents of DAC < contents of location specified by EA.	LT

On some processors, DFC works correctly only on normalized numbers as follows. The comparison has a maximum of three sequential stages:

first the signs, then the exponents, and finally the fractions of the two numbers are compared for equality. If the comparison during any one of these stages reveals an inequality, the results are returned and the instruction ends. Unnormalized numbers are unexpected and produce unexpected results. Other processors actually perform a subtract, resulting in a proper comparison.

#### Note

This instruction also has a register-to-register and an immediate form. See Appendix B for more information.

► DFCM f  
Double Floating Complement  
0 1 1 0 0 0 0 F 0 1 1 0 0 1 0 0

Forms the two's complement of the double precision, floating-point number contained in the specified DAC and normalizes it if necessary. Stores the result in the DAC. An overflow causes a floating-point exception. If no floating-point exception occurs, CBIT is reset to 0. The values of LINK and the condition codes are indeterminate.

If a floating-point exception occurs and bit 7 in the keys contains 1, the instruction sets CBIT to 1. If bit 7 contains 0, the instruction sets CBIT to 1 and causes a floating-point exception fault. For more information, see Chapter 10 of the System Architecture Reference Guide.

► DFD f, address  
Double Floating Divide  
0 1 1 1 1 0 0 F 1 TM\2 SR\3 BR\2  
[ DISPLACEMENT\16 ]

Calculates an effective address, EA. Divides the contents of the specified DAC by the contents of the location specified by EA. Normalizes the quotient if necessary. Stores the result in the DAC. An overflow or divide by zero causes a floating-point exception. If no floating-point exception occurs, CBIT is reset to 0. The values of LINK and the condition codes are indeterminate.

If a floating-point exception occurs and bit 7 in the keys contains 1, the instruction sets CBIT to 1. If bit 7 contains 0, the instruction sets CBIT to 1 and causes a floating-point exception fault. For more information, see Chapter 10 of the System Architecture Reference Guide.

#### Note

This instruction also has a register-to-register and an immediate form. See Appendix B for more information.

► DFL f,address  
 Double Floating Load  
 0 0 0 1 1 0 0 F 1 TM\2 SR\3 BR\2  
 [ DISPLACEMENT\16 ]

Calculates an effective address, EA. Loads the 64-bit contents of the location specified by EA into the specified DAC without normalizing the result. Leaves the values of CBIT, LINK, and the condition codes unchanged.

#### Note

The DFL instruction also has a register-to-register and an immediate form. See Appendix B for more information.

► DFM f,address  
 Double Floating Multiply  
 0 1 0 1 1 0 1 F 1 TM\2 SR\3 BR\2  
 [ DISPLACEMENT\16 ]

Calculates an effective address, EA. Multiplies the 64-bit contents of the location specified by EA by the contents of the specified DAC. Normalizes the result if necessary. Stores the result in the DAC. An overflow causes a floating-point exception. If no floating-point exception occurs, CBIT is reset to 0. The values of LINK and the condition codes are indeterminate.

If a floating-point exception occurs and bit 7 in the keys contains 1, the instruction sets CBIT to 1. If bit 7 contains 0, the instruction sets CBIT to 1 and causes a floating-point exception fault. For more information, see Chapter 10 of the System Architecture Reference Guide.

#### Note

This instruction also has a register-to-register and an immediate form. See Appendix B for more information.

► DFS f,address  
 Double Floating Subtract  
 0 1 0 1 1 0 0 F 1 TM\2 SR\3 BR\2  
 [ DISPLACEMENT\16 ]

Calculates an effective address, EA. Subtracts the 64-bit contents of the location specified by EA from the contents of the specified DAC. Stores the result in the DAC. An overflow causes a floating-point exception. If no floating-point exception occurs, CBIT is reset to 0. The values of LINK and the condition codes are indeterminate.

For 750 and 850 processors, exponent underflow is detected, but exponent overflow is not.

If a floating-point exception occurs and bit 7 in the keys contains 1, the instruction sets CBIT to 1. If bit 7 contains 0, the instruction sets CBIT to 1 and causes a floating-point exception fault. For more information, see Chapter 10 of the System Architecture Reference Guide.

#### Note

The DFS instruction also has a register-to-register and an immediate form. See Appendix B for more information.

► DFST f,address  
Double Floating Point Store  
0 0 1 1 1 0 0 F 1 TM\2 SR\3 BR\2  
DISPLACEMENT\16

Calculates an effective address, EA. Stores the contents of the specified DAC into the location specified by EA. Leaves the values of CBIT, LINK, and the condition codes unchanged.

#### Note

This instruction does not normalize the result before loading it into the specified memory location.

► DH R,address  
Divide Halfword  
1 1 1 0 1 0 DR\3 TM\2 SR\3 BR\2  
[ DISPLACEMENT\16 ]

Calculates an effective address, EA. Divides the 32-bit dividend contained in the specified R by the 16-bit value contained in the location specified by EA. Stores the quotient in bits 1 to 16 of R and the remainder in bits 17 to 32 of R. The sign of the remainder equals the sign of the dividend. If the quotient is less than  $-(2^{15})$  or greater than  $(2^{15})-1$ , an overflow occurs and causes an integer exception. If no integer exception occurs, CBIT is reset to 0. The values of LINK and the condition codes are indeterminate.

If an integer exception occurs and bit 8 in the keys contains 0, the instruction sets CBIT to 1. If bit 8 contains 1, the instruction sets CBIT to 1 and causes an integer exception fault. For more information, see Chapter 10 of the System Architecture Reference Guide.

Note

The DH instruction also has a register-to-register and an immediate form. See Appendix B for more information.

► DH1 r  
Decrement r by 1  
0 1 1 0 0 0 R\3 1 0 1 1 0 0 0

Decrements the contents of r by 1 and stores the result in r. If an overflow occurs, an integer exception occurs. If no integer exception occurs, CBIT is reset to 0. LINK reflects the value of the carry. The condition codes reflect the result of the operation. (See Appendix A.)

If an integer exception occurs and bit 8 of the keys contains 0, the instruction sets CBIT to 1. If bit 8 contains a 1, the instruction sets CBIT to 1 and causes an integer exception fault. See Chapter 10 of the System Architecture Reference Guide for more information.

► DH2 r  
Decrement r by 2  
0 1 1 0 0 0 R\3 1 0 1 1 0 0 1

Decrements the contents of r by 2 and stores the result in r. If an overflow occurs, an integer exception occurs. If no integer exception occurs, CBIT is reset to 0. LINK reflects the value of the carry. The condition codes reflect the result of the operation. (See Appendix A.)

If an integer exception occurs and bit 8 of the keys contains 0, the DH2 instruction sets CBIT to 1. If bit 8 contains a 1, the instruction sets CBIT to 1 and causes an integer exception fault. See Chapter 10 of the System Architecture Reference Guide for more information.

► DM address  
Decrement Memory Fullword  
1 1 0 1 1 0 0 0 0 TM\2 SR\3 BR\2  
DISPLACEMENT\16

Subtracts 1 from the 32-bit integer contained in the specified location and stores the result back in the specified location. Leaves the values of LINK and CBIT unchanged. The condition codes reflect the result of the operation. (See Appendix A.)

► DMH address  
 Decrement Memory Halfword  
 1 1 1 1 1 0 0 0 0 TM\2 SR\3 BR\2  
 DISPLACEMENT\16

Subtracts 1 from the 16-bit integer contained in the specified location and stores the result back in the specified location. Leaves the values of LINK and CBIT unchanged. The condition codes reflect the result of the operation. (See Appendix A.)

► DR1 R  
 Decrement Register by 1  
 0 1 1 0 0 0 R\3 1 0 1 0 1 0 0

Decrements the contents of R by 1 and stores the result in R. An overflow causes an integer exception. If no integer exception occurs, CBIT is reset to 0. LINK contains the value of the borrow bit. The condition codes reflect the result of the operation. (See Appendix A.)

If an integer exception occurs and bit 8 of the keys contains 0, the instruction sets CBIT to 1. If bit 8 contains a 1, the instruction sets CBIT to 1 and causes an integer exception fault. See Chapter 10 of the System Architecture Reference Guide for more information.

► DR2 R  
 Decrement Register by 2  
 0 1 1 0 0 0 R\3 1 0 1 0 1 0 1

Decrements the contents of the specified R by 2 and stores the result in R. An overflow causes an integer exception. If no integer exception occurs, CBIT is reset to 0. LINK contains the value of the borrow bit. The condition codes reflect the result of the operation. (See Appendix A.)

If an integer exception occurs and bit 8 of the keys contains 0, the DR2 instruction sets CBIT to 1. If bit 8 contains a 1, the instruction sets CBIT to 1 and causes an integer exception fault. See Chapter 10 of the System Architecture Reference Guide for more information.

► DRN  
 Double Round From Quad  
 0 1 0 0 0 0 0 0 1 1 0 0 0 0 0 0

Converts the 112-bit value in QAC to a double precision floating-point number. If QAC contains 0, the instruction ends. If bits 50 to 96 of QAC are not zero, or bit 48 of QAC contains 1, the instruction adds the value of bit 49 to that of bit 48 (unbiased round) and clears bits 49 to 96 of QAC to 0. If any other condition exists, no unbiased rounding

occurs, but bits 49 to 96 of QAC are still cleared to 0. After any rounding and clearing occurs, the instruction normalizes the result and loads it into bits 1 to 64 of QAC.

If no floating-point exception occurs, the instruction resets CBIT to 0. The values of LINK and the condition codes are indeterminate.

If a floating-point exception occurs and bit 7 of the keys contains a 1, the instruction sets CBIT to 1. If bit 7 contains a 0, the instruction sets CBIT to 1 and causes a floating-point exception fault. See Chapter 10 of the System Architecture Reference Guide for more information.

#### Note

If DRN is used for any earlier system listed in "About This Book", an unimplemented instruction (UII) fault occurs. (See Chapter 10 of the System Architecture Reference Guide.)



#### DRNM

Double Round From Quad Towards Negative Infinity

1 1 0 0 0 0 0 1 0 1 1 1 1 0 0 1

Converts the 112-bit value in QAC to a double precision floating-point number. If QAC contains 0, or if bits 49 to 96 of QAC contain zeros, the instruction ends. In any other case, the instruction clears bits 49 to 96 to 0, normalizes the result, and places it in bits 1 to 64 of QAC.

The value of CBIT is unchanged. The values of LINK and the condition codes are indeterminate.

#### Note

If DRNM is used for any earlier system listed in "About This Book", an unimplemented instruction (UII) fault occurs. (See Chapter 10 of the System Architecture Reference Guide.)



#### DRNP

Double Round From Quad Towards Positive Infinity

0 1 0 0 0 0 0 0 1 1 0 0 0 0 0 1

Converts the 112-bit value in QAC to a double precision floating point number. If QAC contains 0, or if bits 49 to 96 of QAC contain zeros, the instruction ends. In any other case, the instruction adds 1 to the value contained in bit 48 of QAC, clears bits 49 to 96 to 0, normalizes the result, and places it in bits 1 to 64 of QAC.

If no floating-point exception occurs, the instruction resets CBIT to 0. The values of LINK and the condition codes are indeterminate.

If a floating-point exception occurs and bit 7 of the keys contains a 1, the instruction sets CBIT to 1. If bit 7 contains a 0, the instruction sets CBIT to 1 and causes a floating-point exception fault. See Chapter 10 of the System Architecture Reference Guide for more information.

#### Note

If DRNP is used for any earlier system listed in "About This Book", an unimplemented instruction (UII) fault occurs. (See Chapter 10 of the System Architecture Reference Guide.)

► DRNZ  
Double Round From Quad Towards Zero  
0 1 0 0 0 0 0 0 1 1 0 0 0 0 1 0

Converts the 112-bit value in QAC to a double precision floating-point number. If QAC contains 0, the instruction ends. If bits 49 to 96 of QAC contain zeros and bit 1 contains 1, the instruction adds 1 to the value contained in bit 48 of QAC, clears bits 49 to 96 to 0, normalizes the result, and places it in bits 1 to 64 of QAC. If any other condition exists, no rounding occurs.

If no floating-point exception occurs, the instruction resets CBIT to 0. The values of LINK and the condition codes are indeterminate.

If a floating-point exception occurs and bit 7 of the keys contains a 1, the instruction sets CBIT to 1. If bit 7 contains a 0, the instruction sets CBIT to 1 and causes a floating-point exception fault. See Chapter 10 of the System Architecture Reference Guide for more information.

#### Note

If DRNZ is used for any earlier system listed in "About This Book", an unimplemented instruction (UII) fault occurs. (See Chapter 10 of the System Architecture Reference Guide.)

► E16S  
 Enter 16S Mode  
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1

Sets bits 4 to 6 of the keys to 000. Subsequent S mode instructions may now be interpreted, and 16S address calculations may be used to form effective addresses. Leaves the values of CBIT, LINK, and the condition codes unchanged.

► E32I  
 Enter 32I Mode  
 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0

Sets bits 4 to 6 of the keys to 100. Subsequent I mode instructions may now be interpreted, and 32I address calculations may be used to form effective addresses. Leaves the values of CBIT, LINK, and the condition codes unchanged.

► E32R  
 Enter 32R Mode  
 0 0 0 0 0 0 1 0 0 0 0 0 1 0 1 1

Sets bits 4 to 6 of the keys to 011. Subsequent R mode instructions may now be interpreted, and 32R address calculations may be used to form effective addresses. Leaves the values of CBIT, LINK, and the condition codes unchanged.

► E32S  
 Enter 32S Mode  
 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 1

Sets bits 4 to 6 of the keys to 001. Subsequent S mode instructions may now be interpreted, and 32S address calculations may be used to form effective addresses. Leaves the values of CBIT, LINK, and the condition codes unchanged.

► E64R  
 Enter 64R Mode  
 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 1

Sets bits 4 to 6 of the keys to 010. Subsequent R mode instructions may now be interpreted, and 64R address calculations may be used to form effective addresses. Leaves the values of CBIT, LINK, and the condition codes unchanged.

► E64V  
 Enter 64V Mode  
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0

Sets bits 4 to 6 of the keys to 110. Subsequent V mode instructions may now be interpreted, and 64V address calculations may be used to form effective addresses. Leaves the values of CBIT, LINK, and the condition codes unchanged.

► EAFA far,address  
 Effective Address to FAR  
 0 0 0 0 0 0 1 0 1 1 0 0 F 0 0 0  
 AP\32

Builds a 36-bit EA from the 32-bit address pointer contained in the instruction and loads it into the specified FAR. The AP bit field is processed and loaded into the bit portion of the FAR, for direct access. Indirection uses the bit field in the indirect pointer (if any). Leaves the values of CBIT, LINK, and the condition codes unchanged.

Figure 3-2 shows the format of the EA loaded into the specified FAR.

1	16 17	32 33	36
RING, SEG	WORD #	BIT #	

EA Format for EAFA  
 Figure 3-2

► EALB address  
 Effective Address to LB  
 1 0 0 1 1 0 0 1 0 TM\2 SR\3 BR\2  
 DISPLACEMENT\16

Calculates an effective address, EA, and loads it into LB. Leaves the values of CBIT, LINK, and the condition codes unchanged.

- **EAR R,address**  
 Effective Address to Register  
 1 1 0 0 1 1 DR\3 TM\2 SR\3 BR\2  
 DISPLACEMENT\16

Calculates an effective address, EA. Loads the 32-bit EA into the specified R. Leaves the values of CBIT, LINK, and the condition codes unchanged.

- **EAXB address**  
 Effective Address to XB  
 1 0 1 1 1 0 0 1 0 TM\2 SR\3 BR\2  
 DISPLACEMENT\16

Calculates an effective address, EA, and loads it into XB. Leaves the values of CBIT, LINK, and the condition codes unchanged.

- **EIO address**  
 Execute I/O  
 0 1 1 1 0 0 DR\3 TM\2 SR\3 BR\2  
 DISPLACEMENT\16

Calculates an effective address, EA. Executes bits 17 to 32 of EA as if they were a PIO instruction. If execution is successful, the instruction sets the condition codes as follows:

<u>CC</u>	<u>Meaning</u>
EQ	Successful INA, OTA, or SKS instruction
NE	Unsuccessful INA, OTA, OR SKS; any OCP

Leaves the values of LINK and CBIT unchanged. For more information about I/O operations, see Chapter 11 of the System Architecture Reference Guide.

#### Note

This is a restricted instruction.

► ENB  
 Enable Interrupts  
 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1

Enables interrupts by setting bit 1 of the modals to 1. Inhibits interrupts for one instruction. Leaves the values of CBIT, LINK, and the condition codes unchanged.

Note

This is a restricted instruction.

► ENBL  
 Enable Interrupts (Local)  
 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1

This 850 instruction performs the same actions as ENB, except that it is performed specifically for the local processor. Leaves the values of CBIT, LINK, and the condition codes unchanged.

Note

This is a restricted instruction.

► ENBM  
 Enable Interrupts (Mutual)  
 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0

For the 850, a processor checks the availability of the mutual exclusion lock. If available, the processor releases this lock and enables interrupts. Leaves the values of CBIT, LINK, and the condition codes unchanged.

Note

This is a restricted instruction.

## INSTRUCTION SETS GUIDE

► ENBP  
Enable Interrupts (Process)  
0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0

For the 850, a processor checks the availability of the process exchange lock. If available, the process releases this lock and enables interrupts. Leaves the values of CBIT, LINK, and the condition codes unchanged.

### Note

This is a restricted instruction.

► FA f,address  
 Floating Add  
 0 0 1 1 1 0 1 F 0 TM\2 SR\3 BR\2  
 [ DISPLACEMENT\16 ]

Calculates an effective address, EA. Adds the contents of the specified FAC to the 32-bit contents of the location specified by EA. (See Chapter 6 of the System Architecture Reference Guide.) Stores the result in the FAC. An overflow causes a floating-point exception. If no floating-point exception occurs, CBIT is reset to 0. The values of LINK and the condition codes are indeterminate. If a floating-point exception occurs and bit 7 of the keys contains a 1, the instruction sets CBIT to 1. If bit 7 contains a 0, the instruction sets CBIT to 1 and causes a floating-point exception fault. See Chapter 10 of the System Architecture Reference Guide.

#### Note

This instruction also has a register-to-register and an immediate form. See Appendix B for more information.

► FC f,address  
 Floating Compare  
 0 0 0 1 1 0 1 F 0 TM\2 SR\3 BR\2  
 [ DISPLACEMENT\16 ]

Calculates an effective address, EA. Compares the contents of the specified FAC to the contents of the location specified by EA. Leaves the values of LINK and CBIT unchanged. Sets the condition codes to reflect the outcome of the comparison:

<u>Condition</u>	<u>CC</u>
Contents of FAC > contents of location specified by EA.	GT
Contents of FAC = contents of location specified by EA.	EQ
Contents of FAC < contents of location specified by EA.	LT

On some processors, FC works correctly only on normalized numbers as follows. The comparison has a maximum of three sequential stages: first the signs, then the exponents, and finally the fractions of the two numbers are compared for equality. If the comparison during any one of these stages reveals an inequality, the results are returned and the instruction ends. Unnormalized numbers are unexpected and produce unexpected results. Other processors actually perform a subtract, resulting in a proper comparison.

#### Note

The FC instruction also has a register-to-register and an immediate form. See Appendix B for more information.

► FCDQ  
 Floating Point Convert Double to Quad  
 1 1 0 0 0 0 0 1 0 1 1 1 1 0 0 1

Clears FAC1 to 0. Leaves the values of CBIT, LINK, and the condition codes unchanged.

#### Note

If FCDQ is used for any earlier system listed in "About This Book", an unimplemented instruction (UII) fault occurs. (See Chapter 10 of the System Architecture Reference Guide.)

► FCM f  
 Floating Point Complement  
 0 1 1 0 0 0 0 F 0 1 0 0 0 0 0 0

Forms the two's complement of the contents of the FAC and normalizes the result if necessary. (See Chapter 6 of the System Architecture Reference Guide.) Stores the result in the FAC. An overflow causes a floating-point exception. If no floating-point exception occurs, CBIT is reset to 0. The values of LINK and the condition codes are indeterminate.

If a floating-point exception occurs and bit 7 in the keys contains 1, the instruction sets CBIT to 1. If bit 7 contains 0, the instruction sets CBIT to 1 and causes a floating-point exception fault. For more information, see Chapter 10 of the System Architecture Reference Guide.

► FD f,address  
 Floating Divide  
 0 1 1 1 1 0 0 F 0 TM\2 SR\3 BR\2  
 [ DISPLACEMENT\16 ]

Calculates an effective address, EA. Divides the contents of the specified FAC by the contents of the location specified by EA. (See Chapter 6 of the System Architecture Reference Guide.) Stores the result in the FAC and normalizes if necessary. A divide by 0 or an overflow causes a floating-point exception. If no floating-point exception occurs, CBIT is reset to 0. The values of LINK and the condition codes are indeterminate.

If a floating-point exception occurs and bit 7 in the keys contains 1, the instruction sets CBIT to 1. If bit 7 contains 0, the instruction sets CBIT to 1 and causes a floating-point exception fault. For more information, see Chapter 10 of the System Architecture Reference Guide.

Note

The FD instruction also has a register-to-register and an immediate form. See Appendix B for more information.

► FL f,address  
Floating Load  
0 0 0 1 1 0 0 F 0 TM\2 SR\3 BR\2  
[ DISPLACEMENT\16 ]

Calculates an effective address, EA. Converts the single precision operand to double precision and loads the result into the specified FAC without normalizing it. Leaves the contents of CBIT, LINK, and the condition codes unchanged.

Note

This instruction also has a register-to-register and an immediate form. See Appendix B for more information.

► FLT f,R  
Convert Integer to Floating Point  
0 1 1 0 0 0 R\3 1 0 0 F 1 0 1

Converts the integer contained in R to a floating-point number and stores the result in the specified FAC. The values of CBIT, LINK, and the condition codes are indeterminate.

► FLTH f,r  
Convert Halfword Integer to Floating Point  
0 1 1 0 0 0 R\3 1 0 0 F 0 1 0

Converts the halfword integer contained in r to a floating-point number and stores the result in the specified FAC. The values of CBIT, LINK, and the condition codes are indeterminate.

► FM f,address  
Floating Multiply  
0 1 0 1 1 0 1 F 0 TM\2 SR\3 BR\2  
[ DISPLACEMENT\16 ]

Calculates an effective address, EA. Multiplies the 32-bit contents of the location specified by EA by the contents of the specified FAC. (See Chapter 6 of the System Architecture Reference Guide.) Normalizes the result, if necessary, and stores it in the FAC. An exponent

overflow causes a floating-point exception. If no floating-point exception occurs, CBIT is reset to 0. The values of LINK and the condition codes are indeterminate.

If a floating-point exception occurs and bit 7 in the keys contains 1, the instruction sets CBIT to 1. If bit 7 contains 0, the instruction sets CBIT to 1 and causes a floating-point exception fault. For more information, see Chapter 10 of the System Architecture Reference Guide.

#### Note

The FM instruction also has a register-to-register and an immediate form. See Appendix B for more information.

► FRN f  
Floating Round  
0 1 1 0 0 0 0 F 0 1 0 0 0 1 1 1

This instruction operates on and stores all results in the floating accumulator.

For the 2350 to 9955 II, the following actions occur. If bits 1 to 48 contain 0, then bits 49 to 64 are cleared to 0. If bits 24 and 25 both contain 1, then 1 is added to bit 24, bits 25 to 48 are cleared to 0, and the result is normalized. If bit 25 contains 1 and bits 26 to 48 are not equal to 0, then 1 is added to bit 24, bits 25 to 48 are cleared, and the result is normalized.

For the earlier systems listed in "About This Book", the following actions occur. If bits 1 to 48 contain 0, then bits 49 to 64 are cleared to 0. Otherwise, bit 25 is added to bit 24, bits 25 to 48 are cleared to 0, and the result is normalized.

For all systems, if no floating-point exception occurs, resets CBIT to 0. The values of LINK and the condition codes are indeterminate.

If a floating-point exception occurs and bit 7 of the keys contains a 1, the instruction sets CBIT to 1. If bit 7 contains a 0, the instruction sets CBIT to 1 and causes a floating-point exception fault. See Chapter 10 of the System Architecture Reference Guide for more information.

► FRNM f  
 Floating Point Round Towards Negative Infinity  
 0 1 1 0 0 0 0 F 0 1 1 0 0 1 1 0

Converts the 64-bit value in DAC to a single precision floating-point number. If DAC contains 0, or if bits 25 to 48 of DAC contain zeros, the instruction ends. In any other case, the instruction clears bits 25 to 48 to 0, normalizes the result, and places it in DAC.

If no floating-point exception occurs, the instruction resets CBIT to 0. The values of LINK and the condition codes are indeterminate.

If a floating-point exception occurs and bit 7 of the keys contains a 1, the instruction sets CBIT to 1. If bit 7 contains a 0, the instruction sets CBIT to 1 and causes a floating-point exception fault. See Chapter 10 of the System Architecture Reference Guide for more information.

► FRNP f  
 Floating Point Round Towards Positive Infinity  
 0 1 1 0 0 0 0 F 0 1 1 0 0 1 0 1

Converts the 64-bit value in DAC to a single precision floating-point number. If DAC contains 0, or if bits 25 to 48 of DAC contain zeros, the instruction ends. In any other case, the instruction adds 1 to the value contained in bit 24 of DAC, clears bits 25 to 48 to 0, normalizes the result, and places it in DAC.

If no floating-point exception occurs, the instruction resets CBIT to 0. The values of LINK and the condition codes are indeterminate.

If a floating-point exception occurs and bit 7 of the keys contains a 1, the instruction sets CBIT to 1. If bit 7 contains a 0, the instruction sets CBIT to 1 and causes a floating-point exception fault. See Chapter 10 of the System Architecture Reference Guide for more information.

► FRNZ f  
 Floating Point Round Towards Zero  
 0 1 1 0 0 0 0 F 0 1 1 0 0 1 1 1

Converts the 64-bit value in DAC to a single precision floating-point number. If DAC contains 0, the instruction ends. If bits 25 to 48 of DAC are not zeros and bit 1 contains 1, the instruction adds 1 to the value contained in bit 24 of DAC, clears bits 25 to 48 to 0, normalizes the result, and places it in DAC. If any other condition exists, no rounding occurs.

If no floating-point exception occurs, the instruction resets CBIT to 0. The values of LINK and the condition codes are indeterminate.

If a floating-point exception occurs and bit 7 of the keys contains a 1, FRNZ sets CBIT to 1. If bit 7 contains a 0, the instruction sets CBIT to 1 and causes a floating-point exception fault. See Chapter 10 of the System Architecture Reference Guide for more information.

► FS f,address  
 Floating Subtract  
 0 1 0 1 1 0 0 F 0 TM\2 SR\3 BR\2  
 [ DISPLACEMENT\16 ]

Calculates an effective address, EA. Subtracts the 32-bit contents of the location specified by EA from the contents of the specified FAC. (See Chapter 6 of the System Architecture Reference Guide.) Normalizes the result, if necessary, and stores it in the FAC. An overflow causes a floating-point exception. If no floating-point exception occurs, CBIT is reset to 0. The values of LINK and the condition codes are indeterminate.

If a floating-point exception occurs and bit 7 of the keys contains a 1, the instruction sets CBIT to 1. If bit 7 contains a 0, the instruction sets CBIT to 1 and causes a floating-point exception fault. See Chapter 10 of the System Architecture Reference Guide for more information.

#### Note

This instruction also has a register-to-register and an immediate form. See Appendix B for more information.

► FST f,address  
 Floating Store  
 0 0 1 1 1 0 0 F 0 TM\2 SR\3 BR\2  
 DISPLACEMENT\16

Calculates an effective address, EA. Stores the contents of the specified FAC into the 32-bit location specified by EA. (See Chapter 6 of the System Architecture Reference Guide.) The result is normalized only if rounding is enabled. If the exponent contained in the FAC is too large to be expressed in 8 bits, a floating-point exception (store exception) occurs. If no exception occurs, the instruction resets CBIT to 0. At the end of the instruction, the values of LINK and the condition codes are indeterminate.

If a floating-point exception occurs and bit 7 of the keys contains a 1, the instruction sets CBIT to 1. If bit 7 contains a 0, the instruction sets CBIT to 1 and causes a floating-point exception fault. See Chapter 10 of the System Architecture Reference Guide for more information. In either case, a floating-point exception leaves the contents of the memory location in an indeterminate state.

► HLT  
Halt  
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Halts computer operation. The program counter points to the instruction that would have been executed if execution had not been stopped. The supervisor terminal indicates a halt. Leaves the values of CBIT, LINK, and the condition codes unchanged.

This instruction saves the contents of registers in a memory location specified by the RSAVPTR. The contents of RSAVPTR can be accessed by an LDAR/STAR instruction with address '40037. The registers are saved in their physical order. (See Chapter 9 of the System Architecture Reference Guide for the format of these register files.) The saved register file order is shown in Table 3-3.

Table 3-3  
Order of Saved Registers After HLT

6350, 9750 to 9955 II	2350 to 2755, 9650 and 9655	Earlier Systems*
User Reg Set 3	User Reg Set 1	User Reg Set 2
User Reg Set 4	User Reg Set 2	User Reg Set 1
User Reg Set 1	User Reg Set 3	DMx Reg File
User Reg Set 2	User Reg Set 4	Microcode Reg File
Microcode Reg File, Set 2	User Reg Set 5	
Indirect Reg Set	User Reg Set 6	
Microcode Reg File, Set 1	User Reg Set 7	
DMx Reg File	User Reg Set 8	
	DMx Reg File	
	Microcode Reg File, Set 1	
	Microcode Reg File, Set 2	

\* The earlier systems are listed in "About This Book". Of these, the 850 has two ISPs. For each ISP, the order of saved registers is identical to the order shown for the rest of the 50 Series.

#### Note

This is a restricted instruction.

- I R, address  
Interchange Register and Memory Fullword  
1 0 0 0 0 1 DR\3 TM\2 SR\3 BR\2  
[ DISPLACEMENT\16 ]

Calculates an effective address, EA. Interchanges the 32-bit value contained in the specified R with the 32-bit value contained in the location specified by EA. Leaves the values of CBIT, LINK, and the condition codes unchanged.

#### Note

The I instruction is non-atomic, and, especially for dual-stream processors, cannot be used for spin-locks. In these cases, use the STCD instruction instead.

This instruction also has a register-to-register form. See Appendix B for more information.

- ICBL r  
Interchange Bytes and Clear Left  
0 1 1 0 0 0 R\3 0 1 1 0 1 0 1

Interchanges bits 1 to 8 and bits 9 to 16 of the specified r, then loads 0 into bits 1 to 8 of r. Leaves the values of CBIT, LINK, and the condition codes unchanged.

- ICER r  
Interchange Bytes and Clear Right  
0 1 1 0 0 0 R\3 0 1 1 0 1 1 0

Interchanges bits 1 to 8 and bits 9 to 16 of the specified r, then loads zeros into bits 9 to 16 of r. Leaves the values of CBIT, LINK, and the condition codes unchanged.

- ICHL R  
Interchange Halfwords and Clear Left  
0 1 1 0 0 0 R\3 0 1 1 0 0 0 0

Interchanges the contents of bits 1 to 16 and bits 17 to 32 of the specified R, then loads zeros into bits 1 to 16 of R. Leaves the values of CBIT, LINK, and the condition codes unchanged.

► ICHR R  
Interchange Halfwords and Clear Right  
0 1 1 0 0 0 R\3 0 1 1 0 0 0 1

Interchanges the contents of bits 1 to 16 and bits 17 to 32 of the specified R, then loads zeros into bits 17 to 32 of R. Leaves the values of CBIT, LINK, and the condition codes unchanged.

► ICP R  
Increment C Pointer  
0 1 1 0 0 0 R\3 1 1 1 0 1 1 1

Increments the C language pointer in the specified R by 1 byte. Incrementing the largest offset adds 1 to the segment number. Incrementing the largest segment number with the largest offset generates a pointer to segment 0, offset 0, byte 1. Leaves the CBIT, LINK, and the condition codes unchanged. (For C pointer details, see I Mode in Chapter 1 and 32 I Mode in Appendix B of this guide.)

#### Note

If ICP is used for any earlier system listed in "About This Book", an unimplemented instruction (UII) fault occurs. (See Chapter 10 of the System Architecture Reference Guide.)

► IH r, address  
Interchange r and Memory Halfword  
1 0 1 0 0 1 DR\3 TM\2 SR\3 BR\2  
[ DISPLACEMENT\16 ]

Calculates an effective address, EA. Interchanges the value contained in the specified r with the 16-bit value contained in the location specified by EA. Leaves the values of CBIT, LINK, and the condition codes unchanged.

#### Note

The IH instruction is non-atomic, and, especially for dual-stream processors, cannot be used for spin-locks. In these cases, use the STCH instruction instead.

This instruction also has a register-to-register form. See Appendix B for more information.

► **IH1 r**  
 Increment r by 1  
 0 1 1 0 0 0 R\3 1 0 1 0 1 1 0

Increments the contents of the specified r by 1 and stores the result in r. An overflow causes an integer exception. If no integer exception occurs, CBIT is reset to 0. LINK reflects the state of the carry. The condition codes reflect the result of the operation. (See Appendix A.)

If an integer exception occurs and bit 8 of the keys contains 0, the instruction sets CBIT to 1. If bit 8 contains a 1, the instruction sets CBIT to 1 and causes an integer exception fault. See Chapter 10 of the System Architecture Reference Guide for more information.

► **IH2 r**  
 Increment r by 2  
 0 1 1 0 0 0 R\3 1 0 1 0 1 1 1

Increments the contents of the specified r by 2 and stores the result in r. An overflow causes an integer exception to occur. If no integer exception occurs, CBIT is reset to 0. LINK reflects the state of the carry. The condition codes reflect the result of the operation. (See Appendix A.)

If an integer exception occurs and bit 8 of the keys contains 0, the instruction sets CBIT to 1. If bit 8 contains a 1, the instruction sets CBIT to 1 and causes an integer exception fault. See Chapter 10 of the System Architecture Reference Guide for more information.

► **IM address**  
 Increment Memory Fullword  
 1 0 0 1 1 0 0 0 0 TM\2 SR\3 BR\2  
 DISPLACEMENT\16

Adds 1 to the 32-bit integer contained in the specified location and stores the result back in the specified location. Leaves the values of LINK and CBIT unchanged. The condition codes reflect the result of the operation. (See Appendix A.)

► **TMH address**  
 Increment Memory Halfword  
 1 0 1 1 1 0 0 0 0 TM\2 SR\3 BR\2  
 DISPLACEMENT\16

Adds 1 to the 16-bit integer contained in the specified location and stores the result back in the specified location. Leaves the values of LINK and CBIT unchanged. The condition codes reflect the result of the operation. (See Appendix A.)

► **INBC address**  
 Interrupt Notify Beginning, Clear Active Interrupt  
 0 0 0 0 0 0 1 0 1 0 0 0 1 1 1 1  
 AP\32

Notifies a semaphore at the specified address during phantom interrupt code. Restores the state of the interrupted process by loading bits 1 to 16 of PB, bits 17 to 32 of the program counter, and the keys from microcode temporary registers PSWPB and PSWKEYS. Places the notified process at the beginning of the appropriate priority level queue. Issues a CAI pulse to clear the currently active interrupt, and enables interrupts.

The values of CBIT, LINK, and the condition codes are indeterminate. A process exchange will occur if the notified process is of a higher priority than the interrupted process. See Chapter 9 of the System Architecture Reference Guide for more information.

#### Note

INBC is a restricted instruction.

This instruction is normally used to transfer from phantom interrupt code to an interrupt process. See Chapter 10 of the System Architecture Reference Guide for more information.

► **INBN address**  
 Interrupt Notify Beginning  
 0 0 0 0 0 0 1 0 1 0 0 0 1 1 0 1  
 AP\32

Notifies a semaphore at the specified address during phantom interrupt code. Restores the state of the interrupted process by loading bits 1 to 16 of PB, bits 17 to 32 of the program counter, and the keys from microcode temporary registers PSWPB and PSWKEYS. Places the notified process at the beginning of the appropriate priority level queue, and enables interrupts. Does not issue a CAI pulse to clear the currently active interrupt.

The values of CBIT, LINK, and the condition codes are indeterminate. A process exchange will occur if the notified process is of a higher priority than the interrupted process. See Chapter 9 of the System Architecture Reference Guide for more information.

#### Note

INBN is a restricted instruction.

This instruction is normally used to transfer from phantom interrupt code to an interrupt process. See Chapter 10 of the System Architecture Reference Guide for more information.

► INEC address  
Interrupt Notify End, Clear Active Interrupt  
0 0 0 0 0 1 0 1 0 0 0 1 1 1 0  
AP\32

Notifies a semaphore at the specified address during phantom interrupt code. Restores the state of the interrupted process by loading bits 1 to 16 of PB, bits 17 to 32 of the program counter, and the keys from microcode temporary registers PSWPB and PSWKEYS. Places the notified process at the end of the appropriate priority level queue. Issues a CAI pulse to clear the currently active interrupt, and enables interrupts.

The values of CBIT, LINK and the condition codes are indeterminate. A process exchange will occur if the notified process is of a higher priority than the interrupted process. See Chapter 9 of the System Architecture Reference Guide for more information.

#### Note

INEC is a restricted instruction.

This instruction is normally used to transfer from phantom interrupt code to an interrupt process. See Chapter 10 of the System Architecture Reference Guide for more information.

► INEN address  
Interrupt Notify End  
0 0 0 0 0 1 0 1 0 0 0 1 1 0 0  
AP\32

Notifies a semaphore at the specified address during phantom interrupt code. Restores the state of the interrupted process by loading bits 1 to 16 of PB, bits 17 to 32 of the program counter, and the keys from microcode temporary registers PSWPB and PSWKEYS. Places the notified

process at the end of the appropriate priority level queue, and enables interrupts. Does not issue a CAI pulse to clear the currently active interrupt.

The values of CBIT, LINK, and the condition codes are indeterminate. A process exchange will occur if the notified process is of a higher priority than the interrupted process. See Chapter 9 of the System Architecture Reference Guide for more information.

#### Note

This is a restricted instruction.

This instruction is normally used to transfer from phantom interrupt code to an interrupt process. See Chapter 10 of the System Architecture Reference Guide for more information.

► INH  
Inhibit Interrupts  
0 0 0 0 0 0 1 0 0 0 0 0 0 0 1

Inhibits interrupts by resetting bit 1 of the modals to 0. Inhibits interrupts until an enable interrupts instruction executes. The processor ignores any interrupt requests that are made over the I/O bus. This instruction takes effect immediately. Leaves the values of CBIT, LINK, and the condition codes unchanged.

#### Note

This is a restricted instruction.

► INHL  
Inhibit Interrupts (Local)  
0 0 0 0 0 0 1 0 0 0 0 0 0 0 1

This 850 instruction performs the same actions as INH does. Leaves the values of CBIT, LINK, and the condition codes unchanged.

#### Note

INHL is a restricted instruction.

► **INHM**  
 Inhibit Interrupts (Mutual)  
 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0

For the 850, a processor checks the availability of the mutual exclusion lock. If available, the processor sets this lock and inhibits interrupts. Otherwise, it waits for the lock to be released by the other processor and then sets the lock and inhibits interrupts. Leaves the values of CBIT, LINK, and the condition codes unchanged.

Note

This is a restricted instruction.

► **INHP**  
 Inhibit Interrupts (Process)  
 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0

For the 850, a processor checks the availability of the process exchange lock. If available, the processor sets it and inhibits interrupts. Otherwise, it waits for the lock to be released by the other processor, and then sets the lock and inhibits interrupts. It also inhibits interrupts in the local processor. Leaves the values of CBIT, LINK, and the condition codes unchanged.

Note

This is a restricted instruction.

► **INK r**  
 Input Keys  
 0 1 1 0 0 0 R\3 0 1 1 1 0 0 0

Loads the contents of the I mode keys into the specified r. Leaves the values of CBIT, LINK, and the condition codes unchanged. Reads the low-order 8 bits of the S register along with the high-order 8 bits of the keys register.

► **INT f,R**  
 Convert Floating Point to Integer  
 0 1 1 0 0 0 R\3 1 0 0 F 0 1 1

Converts the double precision floating-point number contained in the specified floating accumulator to a 32-bit integer and stores the result in R. Ignores the fractional part of the floating-point number. For example, +4.5 is converted to +4 and -4.5 is converted to -4.

Overflow occurs if the value in the floating accumulator is less than  $-2^{*}31$  or greater than  $(2^{*}31)-1$ . An overflow causes a floating-point exception. If no floating-point exception occurs, CBIT is reset to 0. The values of LINK and the condition codes are indeterminate.

If a floating-point exception occurs and bit 7 of the keys contains a 1, the instruction sets CBIT to 1. If bit 7 contains a 0, the instruction sets CBIT to 1 and causes a floating-point exception fault. See Chapter 10 of the System Architecture Reference Guide for more information.

► INTF f,r  
Convert Floating Point Number to Halfword Integer  
0 1 1 0 0 0 R\3 1 0 0 F 0 0 1

Converts the double precision floating-point number contained in the specified floating accumulator to an integer and stores the result in r. Ignores the fractional portion of the floating-point number. For example, +4.5 is converted to +4 and -4.5 is converted to -4. Overflow occurs if the value in the floating accumulator is less than  $-2^{*}15$  or greater than  $(2^{*}15)-1$ . An overflow causes a floating-point exception. If no floating-point exception occurs, CBIT is reset to 0.

At the end of this instruction, the contents of R bits 17 to 32 are indeterminate. The values of LINK and the condition codes are indeterminate.

If a floating-point exception occurs and bit 7 in the keys contains 1, the instruction sets CBIT to 1. If bit 7 contains 0, the instruction sets CBIT to 1 and causes a floating-point exception fault. For more information, see Chapter 10 of the System Architecture Reference Guide.

► IR1 R  
Increment Register by 1  
0 1 1 0 0 0 R\3 1 0 1 0 0 1 0

Increments the contents of the specified R by 1 and stores the result in R. An overflow causes an integer exception fault. If no integer exception occurs, CBIT is reset to 0. LINK contains the carry-out bit. The condition codes reflect the result of the operation. (See Appendix A.)

If an integer exception occurs and bit 8 in the keys contains 0, the IR1 instruction sets CBIT to 1. If bit 8 contains 1, the instruction sets CBIT to 1 and causes an integer exception fault. (See Chapter 10 of the System Architecture Reference Guide.)

► IR2 R  
Increment Register by 2  
0 1 1 0 0 0 R\3 1 0 1 0 0 1 1

Increments the contents of the specified R by 2 and stores the result in R. An overflow causes an integer exception fault. If no integer exception occurs, CBIT is reset to 0. LINK contains the carry-out bit. The condition code contains the result of the operation. (See Appendix A.)

If an integer exception occurs and bit 8 in the keys contains 0, the instruction sets CBIT to 1. If bit 8 contains 1, the instruction sets CBIT to 1 and causes an integer exception fault. For more information, see Chapter 10 of the System Architecture Reference Guide.

► IRB r  
Interchange r Bytes  
0 1 1 0 0 0 R\3 0 1 1 0 1 0 0

Interchanges bits 1 to 8 and bits 9 to 16 of the specified r. Leaves the values of CBIT, LINK, and the condition codes unchanged.

► IRH R  
Interchange Register Halves  
0 1 1 0 0 0 R\3 0 1 0 1 1 1 1

Interchanges the contents of bits 1 to 16 and bits 17 to 32 of the specified R. Leaves the values of CBIT, LINK, and the condition codes unchanged.

► IRTC  
Interrupt Return, Clear Active Interrupt  
0 0 0 0 0 0 0 1 1 0 0 0 0 0 1 1

Returns from an interrupt. Restores the state existing before the interrupt by loading bits 1 to 16 of PB, bits 17 to 32 of the program counter, and the keys from the values saved in microcode temporary registers PSWPB and PSWKEYS. Issues a CAI pulse to clear the currently active interrupt, and enables interrupts.

#### Note

IRTC is a restricted instruction.

► IRTN  
 Interrupt Return  
 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 1

Returns from an interrupt. Restores the state existing before the interrupt by loading bits 1 to 16 of PB, bits 17 to 32 of the program counter, and the keys from the values saved in microcode temporary registers PSWPB and PSWKEYS, and enables interrupts. Does not issue a CAI pulse to clear the currently active interrupt.

#### Note

This is a restricted instruction.

► ITLB  
 Invalidate STLB Entry  
 0 0 0 0 0 0 0 0 1 1 0 0 0 1 1 0 1

Invalidates the STLB entry that corresponds to the virtual address contained in GR2. The values of CBIT, LINK, and the condition codes are indeterminate. You must execute this instruction whenever you change the page table entry for the given address.

If you change an SDW or DTAR (explained in Chapter 4 of the System Architecture Reference Guide), you usually have to invalidate the entire STLB by issuing the instruction PTLB. A 0 in the segment number portion of GR2 invalidates the IOTLB entry corresponding to the address specified by GR2.

#### Note

This is a restricted instruction.

► JMP address  
 Jump  
 1 0 1 1 1 0 0 0 1 TM\2 SR\3 BR\2  
 DISPLACEMENT\16

Calculates an effective address, EA, and loads it into the program counter. Leaves the values of CBIT, LINK, and the condition codes unchanged.

► JSR r,address  
 Jump to Subroutine  
 1 1 1 0 1 1 DR\3 TM\2 SR\3 BR\2  
 DISPLACEMENT\16

Calculates an effective address, EA. Saves the 16-bit halfword number position of the return address in the specified r. Loads the program counter with the current segment location specified by bits 17 to 32 of the EA. Leaves the values of CBIT, LINK, and the condition codes unchanged.

#### Note

This instruction is useful for calling routines within the current segment only.

► JSXB address  
 Jump and Save in XB  
 1 1 0 1 1 0 0 0 1 TM\2 SR\3 BR\2  
 DISPLACEMENT\16

Calculates an effective address, EA. Loads the contents of the program counter into XB. Loads EA into the program counter. Leaves the values of CBIT, LINK, and the condition codes unchanged.

#### Note

JSXB can make subroutine calls outside the current segment as well as within.

► L R, address  
 Load Full Word  
 0 0 0 0 0 1 DR\3 TM\2 SR\3 BR\2  
 [ DISPLACEMENT\16 ]

Calculates an effective address, EA. Loads EA into the specified R. Leaves the values of CBIT, LINK, and the condition codes unchanged.

#### Note

This instruction also has a register-to-register and an immediate form. See Appendix B for more information.

► LOC r, address  
 Load C Character  
 1 0 0 1 0 1 DR\3 TM\2 SR\3 BR\2  
 DISPLACEMENT\16

Calculates a C language pointer and uses it to load a single character into bits 9 to 16 of the specified r. If bit 4 of the C pointer contains 0, bits 1 to 8 of the location contain the character to be loaded; if bit 4 of the pointer contains 1, bits 9 to 16 of the location contain the character.

Clears bits 1 to 8 of r, but leaves bits 17 to 32 of R unchanged. Sets the condition code EQ to 1 (indicating equal to 0) when 0 is loaded; resets EQ to 0 (indicating not equal to zero) for all other characters. The state of the LT condition code is indeterminate. Testing the results should be done using either BCEQ or BCNE branches only. Leaves the values of CBIT and LINK unchanged.

#### Note

The LOC instruction is valid only for general register relative and indirect forms of address formation. Other forms of address formation (including indexing) do not reliably generate the C language pointer.

In particular, do not use the register-to-register or immediate form with the LOC instruction because it would be interpreted as a CCP instruction. (LOC and CCP share the same opcode, but CCP uses the register-to-register form; the immediate form of CCP is undefined, but the preferred implementation is a UII (unimplemented instruction).)

Direct addressing, however, will obtain the first byte (of two) pointed to by the effective address. This assumes that the base register used was loaded with a conventional 32-bit IP with the E bit reset.

If LOC is used for any earlier system listed in "About This Book", a UII fault occurs. (See Chapter 10 of the System Architecture Reference Guide.)

► LCEQ r  
Load Register on Condition Code EQ  
0 1 1 0 0 0 R\3 1 1 0 1 0 1 1

If the condition codes reflect an equal to 0 condition, the instruction loads the specified r with a 1. If they reflect a not equal to 0 condition, the instruction loads r with a 0. Leaves the values of CBIT, LINK, and the condition codes unchanged.

► LOGE r  
Load Register on Condition Code GE  
0 1 1 0 0 0 R\3 1 1 0 1 1 0 0

If the condition codes reflect a greater than or equal to 0 condition, the instruction loads the specified r with a 1. If they reflect a less than 0 condition, the instruction loads r with a 0. Leaves the values of CBIT, LINK, and the condition codes unchanged.

► LOGT r  
Load Register on Condition Code GT  
0 1 1 0 0 0 R\3 1 1 0 1 1 0 1

If the condition codes reflect a greater than 0 condition, the instruction loads the specified r with a 1. If they reflect a less than or equal to 0 condition, the instruction loads r with a 0. Leaves the values of CBIT, LINK, and the condition codes unchanged.

► LCLE r  
Load Register on Condition Code LE  
0 1 1 0 0 0 R\3 1 1 0 1 0 0 1

If the condition codes reflect a less than or equal to 0 condition, the instruction loads the specified r with a 1. If they reflect a greater than 0 condition, the instruction loads r with a 0. Leaves the values of CBIT, LINK, and the condition codes unchanged.

► **LCLT r**  
 Load Register on Condition Code LT  
 0 1 1 0 0 0 R\3 1 1 0 1 0 0 0

If the condition codes reflect a less than 0 condition, the instruction loads the specified r with a 1. If they reflect a greater than or equal to 0 condition, the instruction loads r with a 0. Leaves the values of CBIT, LINK, and the condition codes unchanged.

► **LCNE r**  
 Load Register on Condition Code NE  
 0 1 1 0 0 0 R\3 1 1 0 1 0 1 0

If the condition codes reflect a not equal to 0 condition, the instruction loads the specified r with a 1. If they reflect an equal to 0 condition, the instruction loads r with a 0. Leaves the values of CBIT, LINK, and the condition codes unchanged.

► **LDAR R,address**  
 Load Addressed Register  
 1 0 0 1 0 0 DR\3 TM\2 SR\3 BR\2  
 DISPLACEMENT\16

Calculates a 32-bit (1-word) effective address, EA. Loads the specified R with the contents of the register file location specified by the offset portion of EA. Bit 2 and bit 12 of the offset portion of EA determine the actions of this instruction.

<u>Bit 2</u>	<u>Bit 12</u>	<u>Action</u>
1*	----	Ignore bits 1 and 3 to 9. The offset portion of EA specifies an absolute register number from 0 to '377.
0*	1	Bits 13 to 16 of the offset portion of EA specify one of the registers '20 to '37 in the current register set.
0	0	Bits 13 to 16 of the offset portion of EA specify one of the registers 0 to '17 in the current register set.

\*This is a restricted instruction.

Leaves the values of CBIT and LINK unchanged; the values of the condition codes are indeterminate. See Chapter 9 of the System Architecture Reference Guide for more information about register sets.

Note

If the current ring is not 0 and EA is outside the range of 0 to '17, inclusive, any access causes an RXM violation.

► LDC flr,r  
Load Character  
0 1 1 0 0 0 R\3 1 1 1 FLR 0 1 0

If the contents of the specified FLR are nonzero, the instruction fetches the single character pointed to by the appropriate FAR and loads it into bits 9 to 16 of r. When the FAR's bit field contains 0, it specifies the left byte (bits 1 to 8) of the 16-bit addressed quantity; when the bit field contains 8, the right byte (bits 9 to 16) is specified. This instruction loads zeros into bits 1 to 8 of r. Updates the contents of the FAR by 8 (one byte) so that they point to the next character. Decrements the contents of the specified FLR by 1. Sets the condition codes to NE. Leaves the values of CBIT and LINK unchanged.

If the contents of the specified FLR are 0, the instruction sets the condition codes to EQ.

Note

This instruction uses FAR0 when FLR0 is specified, and FAR1 when FLR1 is specified.

► LEQ R  
Load Register on Equal to 0  
0 1 1 0 0 0 R\3 0 0 0 0 0 1 1

If the contents of the specified R are equal to 0, the instruction loads r with a 1. If not equal to 0, the instruction loads r with a 0. Leaves the values of LINK and CBIT unchanged. The condition codes reflect the result of the comparison. (See Appendix A.)

► LF r  
Logic Set False  
0 1 1 0 0 0 R\3 0 0 0 1 1 1 0

Loads the specified r with 0. Leaves the values of LINK and CBIT unchanged. The values of the condition codes are indeterminate.

► LFEQ f,r  
 Load Register on Floating Accumulator Equal to 0  
 0 1 1 0 0 0 R\3 0 0 1 F 0 1 1

If the contents of the specified floating accumulator are equal to 0, the instruction loads the specified r with a 1; if not equal to 0, the instruction loads r with a 0. Leaves the values of LINK and CBIT unchanged. The condition codes reflect the result of the comparison. (See Appendix A.)

LFEQ works correctly only on normalized or nearly normalized numbers, because it checks fraction bits 1 to 32 only for equal to 0 and less than 0. (See the System Architecture Reference Guide, Chapter 6.)

► LFGE f,r  
 Load Register on Floating Accumulator Greater Than or Equal to 0  
 0 1 1 0 0 0 R\3 0 0 1 F 1 0 0

If the contents of the specified floating accumulator are greater than or equal to 0, the instruction loads the specified r with a 1; if less than 0, the instruction loads r with a 0. Leaves the values of LINK and CBIT unchanged. The condition codes reflect the result of the comparison. (See Appendix A.)

LFGE works correctly only on normalized or nearly normalized numbers, because it checks the first 32 fraction bits only for equal to zero and less than zero. (See Chapter 6 in the System Architecture Reference Guide.)

► LFGT f,r  
 Load Register on Floating Accumulator Greater Than 0  
 0 1 1 0 0 0 R\3 0 0 1 F 1 0 1

If the contents of the specified floating accumulator are greater than 0, the instruction loads the specified r with a 1; if less than or equal to 0, the instruction loads r with a 0. Leaves the values of LINK and CBIT unchanged. The condition codes reflect the result of the comparison. (See Appendix A.)

LFGT works correctly only on normalized or nearly normalized numbers, because it checks the first 32 fraction bits only for equal to zero and less than zero. (See Chapter 6 in the System Architecture Reference Guide.)

► **LFLE f,r**  
 Load Register on Floating Accumulator Less Than or Equal to 0  
 0 1 1 0 0 0 R\3 0 0 1 F 0 0 1

If the contents of the specified floating accumulator are less than or equal to 0, the instruction loads the specified r with a 1; if greater than 0, the instruction loads r with a 0. Leaves the values of LINK and CBIT unchanged. The condition codes reflect the result of the comparison. (See Appendix A.)

LFLE works correctly only on normalized or nearly normalized numbers, because it checks the first 32 fraction bits only for equal to zero and less than zero. (See Chapter 6 in the System Architecture Reference Guide.)

► **LFLI flr,data**  
 Load FLR Immediate  
 0 0 0 0 0 0 1 0 1 1 0 0 F 0 1 1  
 INTEGER\16

Loads the 16-bit, unsigned integer contained in bits 17 to 32 (the second halfword) of the instruction into the specified FLR. Clears the upper bits of the FLR. Leaves the values of CBIT, LINK, the condition codes, and the associated FAR unchanged.

► **LFLT f,r**  
 Load Register on Floating Accumulator Less Than 0  
 0 1 1 0 0 0 R\3 0 0 1 F 0 0 0

If the contents of the specified floating accumulator are less than 0, the instruction loads the specified r with a 1; if greater than or equal to 0, the instruction loads r with a 0. Leaves the values of LINK and CBIT unchanged. The condition codes reflect the result of the comparison. (See Appendix A.)

LFLT works correctly only on normalized or nearly normalized numbers, because it checks the first 32 fraction bits only for equal to zero and less than zero. (See Chapter 6 in the System Architecture Reference Guide.)

► **LFNE f,r**  
 Load Register on Floating Accumulator Not Equal to 0  
 0 1 1 0 0 0 R\3 0 0 1 F 0 1 0

If the contents of the specified floating accumulator are not equal to 0, LFNE loads the specified r with a 1; if equal to 0, LFNE loads r with a 0. Leaves the values of LINK and CBIT unchanged. The condition codes reflect the result of the comparison. (See Appendix A.)

LFNE works correctly only on normalized or nearly normalized numbers, because it checks the first 32 fraction bits only for equal to zero and less than zero. (See Chapter 6 in the System Architecture Reference Guide.)

► LGE R  
Load Register on Greater Than or Equal to 0  
0 1 1 0 0 0 R\3 0 0 0 0 1 0 0

If the contents of the specified R are greater than or equal to 0, the instruction loads r with a 1; if less than 0, the instruction loads r with a 0. Leaves the values of LINK and CBIT unchanged. The condition codes reflect the result of the comparison. (See Appendix A.)

► LGT R  
Load Register on Greater Than 0  
0 1 1 0 0 0 R\3 0 0 0 0 1 0 1

If the contents of the specified R are greater than 0, the instruction loads r with a 1; if less than or equal to 0, the instruction loads r with a 0. Leaves the values of LINK and CBIT unchanged. The condition codes reflect the result of the comparison. (See Appendix A.)

► LH r, address  
Load Halfword  
0 0 1 0 0 1 DR\3 TM\2 SR\3 BR\2  
[ DISPLACEMENT\16 ]

Calculates an effective address, EA. Loads the 16-bit contents contained in the location specified by EA into r. Leaves the values of CBIT, LINK, and the condition codes unchanged.

#### Note

LH also has a register-to-register and an immediate form. (See Appendix B.)

► LHEQ r  
Load r on EQ  
0 1 1 0 0 0 R\3 0 0 0 0 1 0 1 1

If the contents of the specified r are equal to 0, the instruction loads r with a 1; if not equal to 0, the instruction loads r with a 0. Leaves the values of LINK and CBIT unchanged. The condition codes reflect the result of the comparison. (See Appendix A.)

► LHGE r  
 Load r on GE  
 0 1 1 0 0 0 R\3 0 0 0 0 1 0 0

If the contents of the specified r are greater than or equal to 0, the instruction loads r with a 1; if less than 0, the instruction loads r with a 0. Leaves the values of LINK and CBIT unchanged. The condition codes reflect the result of the comparison. (See Appendix A.)

► LHGT r  
 Load r on GT  
 0 1 1 0 0 0 R\3 0 0 0 1 1 0 1

If the contents of the specified r are greater than 0, the instruction loads r with a 1; if less than or equal to 0, the instruction loads r with a 0. Leaves the values of LINK and CBIT unchanged. The condition codes reflect the result of the comparison. (See Appendix A.)

► LHL1 r,address  
 Load Halfword Shifted Left by 1  
 0 0 0 1 0 0 DR\3 TM\2 SR\3 BR\2  
 [ DISPLACEMENT\16 ]

Calculates an effective address, EA. Shifts the contents of the location specified by EA left one bit and stores the result in the specified r. (Shifts zero into the vacated bit.) Leaves the values of CBIT, LINK, and the condition codes unchanged.

#### Note

LHL1 also has a register-to-register form. (See Appendix B.)

► LHL2 r,address  
 Load Halfword Shifted Left by 2  
 0 0 1 1 0 0 DR\3 TM\2 SR\3 BR\2  
 [ DISPLACEMENT\16 ]

Calculates an effective address, EA. Shifts the 16-bit contents of the location specified by EA left two bits and stores the result in the specified r. (Shifts zeros into the vacated bits.) Leaves the values of CBIT, LINK, and the condition codes unchanged.

#### Note

LHL2 also has a register-to-register form. (See Appendix B.)

► **LHL3** r,address  
 Load Halfword Shifted Left by 3  
 0 1 1 1 0 1 DR\3 TM\2 SR\3 ER\2  
 [ DISPLACEMENT\16 ]

Calculates an effective address, EA. Shifts the 16-bit contents of the location specified by EA left three bits and stores the result in the specified r. (Shifts zeros into the vacated bits.) Leaves the values of CBIT, LINK, and the condition codes unchanged.

#### Note

LHL3 also has a register-to-register form. (See Appendix B.)

If LHL3 is used for any earlier system listed in "About This Book", an unimplemented instruction (UII) fault occurs. (See Chapter 10 of the System Architecture Reference Guide.)

► **LHLE** r  
 Load r on LE  
 0 1 1 0 0 0 R\3 0 0 0 1 0 0 1

If the contents of the specified r are less than or equal to 0, the instruction loads r with a 1; if greater than 0, the instruction loads r with 0. Leaves the values of LINK and CBIT unchanged. The condition codes reflect the result of the comparison. (See Appendix A.)

► **LHLT** r  
 Load r on LT  
 0 1 1 0 0 0 R\3 0 0 0 0 0 0 0

If the contents of the specified r are less than 0, the instruction loads r with a 1; if greater than or equal to 0, loads r with a 0. Leaves the values of LINK and CBIT unchanged. The condition codes reflect the result of the comparison. (See Appendix A.)

► **LHNE** r  
 Load r on NE  
 0 1 1 0 0 0 R\3 0 0 0 1 0 1 0

If the contents of the specified r are not equal to 0, the instruction loads r with a 1; if equal to 0, the instruction loads r with a 0. Leaves the values of LINK and CBIT unchanged. The condition codes reflect the result of the comparison. (See Appendix A.)

► LIOT address  
 Load IOTLB  
 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0  
 AP\32

Loads a specified IOTLB entry. Table 3-4 shows the contents of the LIOT entry and the origin of the information. The values of CBIT, LINK, and the condition codes are indeterminate.

Table 3-4  
 LIOT Data

Origin	Description
AP in LIOT	Virtual address in I/O segment (calculated from the EA).
Page table	Physical address (translation of the virtual address) obtained from I/O segment. If the fault bit is set to 1, a page fault occurs.
GR2 register	Target virtual address. This is the segment number and page number of the virtual address that will be used by procedures accessing this information. This is used to help invalidate the proper locations in the cache. The segment number and the low-order 10 bits (offset number in the page) are ignored.

#### Note

This is a restricted instruction.

► LIP R,address  
 Load Indirect Pointer  
 1 1 0 1 0 1 DR\3 TM\2 SR\3 BR\2  
 [ DISPLACEMENT\16 ]

Calculates an effective address, EA. Loads the value contained in the location specified by EA into the specified R. Checks these contents for a pointer fault.

This pointer fault is generated when the contents of the memory location to be loaded into the specified R contain a pointer fault (bit 1 contains 1).

If this pointer fault occurs, the pointer to the memory location is saved in FADDR (SB + 11) as well as bits 1 to 16 of the contents of that memory location FCODEH (SB + 10). After completion of the fault handling mechanism, the instruction can be re-executed. (See Chapter 10 of the System Architecture Reference Guide.)

Leaves the values of CBIT, LINK, and the condition codes unchanged.

#### Note

LIP should weaken the ring field against the ring field of the effective address. This is not done on some current processors, but will be done on all future processors.

If LIP is used for any earlier system listed in "About This Book", an unimplemented instruction (UII) fault occurs. (See Chapter 10 of the System Architecture Reference Guide.)

► LLE R  
Load Register on Less Than or Equal to 0  
0 1 1 0 0 0 R\3 0 0 0 0 0 0 1

If the contents of the specified R are less than or equal to 0, the instruction loads r with a 1. If the contents of R are greater than 0, the instruction loads r with a 0. Leaves the values of LINK and CBIT unchanged. The condition codes reflect the result of the comparison. (See Appendix A.)

► LLT R  
Load Register on Less Than 0  
0 1 1 0 0 0 R\3 0 0 0 0 0 0 0

If the contents of the specified R are less than 0, the instruction loads r with a 1. If the contents of R are greater than or equal to 0, the instruction loads r with a 0. Leaves the values of LINK and CBIT unchanged. The condition codes reflect the result of the comparison. (See Appendix A.)

► LNE R  
Load Register on Not Equal to 0  
0 1 1 0 0 0 R\3 0 0 0 0 0 1 0

If the contents of the specified R are not equal to 0, the instruction loads r with a 1; if equal to 0, the instruction loads r with a 0. Leaves the values of LINK and CBIT unchanged. The condition codes reflect the result of the comparison. (See Appendix A.)

► **LPID**  
 Load Process ID  
 0 0 0 0 0 0 0 1 1 0 0 0 1 1 1 1

Loads the process ID from bits 1 to 10 of GR2 into RPID (the process ID register, which contains the 10 most significant bits of the user's address space). Leaves the values of CBIT, LINK, and the condition codes unchanged.

The RPID data is used to update the process ID field of an STLB entry as required. This RPID data is later used during subsequent memory accesses to verify that STLB data is still valid (STLB hit) or not (STLB miss). This register is for internal machine operation, and should not normally be modified by the user.

#### Note

LPID is a restricted instruction.

► **LPSW address**  
 Load PSW  
 0 0 0 0 0 0 0 1 1 1 0 0 1 0 0 1  
 AP\32

Changes the status of the processor by loading new values into the program counter, keys, and modals. Inhibits interrupts for one instruction.

Addresses a 64-bit (4-halfword) block at the specified location. The block has the following format.

<u>Offset in Block</u>	<u>Contents</u>
1 to 2	New program counter (ring, segment, offset numbers)
3	New keys
4	New modals

Loads the program counter and keys of the currently running process with the contents of the first three offsets (bits 1 to 48), then loads the processor modals with the contents of the fourth offset (bits 49 to 64).

The new value of bit 15 in the keys, the in-dispatch bit, can temporarily halt execution of the current process. This bit is altered by software only during a cold or a warm start. If bit 15 is 0, the currently executing process will continue to execute, but at a location defined by the new value of the program counter. If bit 15 is 1, the

processor enters the dispatcher and dispatches the ready process with the highest priority. When execution resumes for the process that was temporarily halted, execution resumes at the point defined by the value of the new program counter.

Regardless of the value of bit 15, the new value of the modals takes effect immediately, since the modals are associated with the processor, not the process.

The LPSW instruction loads the 64 bits (four halfwords) of the register set that the STLR instruction cannot correctly load. STLR does not update the separate hardware registers the processor uses to maintain duplicate information for optimization. Never use the LPSW instruction to change bits 9 to 11 of the modals. These bits specify the current user register set. This means that if you do not know the current value of these bits, you must do the following each time you want to execute an LPSW:

1. Inhibit interrupts.
2. Read the current values of modal bits 9 to 11 with an LLDR '24 instruction.
3. Mask the old values of the modal bits into the new information.
4. Load the new information into the modals with an LPSW.

For the two common uses of LPSW, you do not have to perform this sequence, since the values of modal bits 9 to 11 are predictable. When you use LPSW after a Master Clear to turn on processor exchange mode, bits 9 to 11 are 010 because the processor is always initialized to register set 2. When you use LPSW to return from a fault, check, or interrupt, simply reload the values stored by the break because these values are still correct.

You should not use LPSW to set bits 16 (the save-done bit) or 15 (the in-dispatcher bit) of the keys, unless you are merely loading status following a fault, check, or interrupt. When issuing LPSW after a Master Clear, make sure you load zeros into both of these bits.

#### Note

This is a restricted instruction.

► **LT r**  
 Logic Set True  
 0 1 1 0 0 0 R\3 0 0 0 1 1 1 1

Loads the specified r with 1. Leaves the values of LINK and CBIT unchanged. The values of the condition codes are indeterminate.

► M R,address  
 Multiply Fullword  
 1 0 0 0 1 0 DR\3 TM\2 SR\3 BR\2  
 [ DISPLACEMENT\16 ]

Calculates an effective address, EA. Multiplies the 32-bit value contained in the location specified by EA by the 32-bit value contained in the specified R. Stores the 64-bit result in the specified R and R+1. The least significant bit of the result is contained in bit 32 of R+1. The 150/250, 450/550/250-II, I450-II, and 2250 processors leave the CBIT and LINK unchanged. The other 50 Series processors reset the value of the CBIT to 0 and leave the value of LINK indeterminate. For all 50 Series processors, the condition codes are unchanged. This instruction cannot cause an overflow or generate an integer exception.

#### Note

R must be an even numbered register.

This instruction also has a register-to-register and an immediate form. See Appendix B for more information.

► MH r,address  
 Multiply Halfword  
 1 0 1 0 1 0 R\3 TM\2 SR\3 BR\2  
 [ DISPLACEMENT\16 ]

Calculates an effective address, EA. Multiplies the 16-bit value contained in the location specified by EA by the 16-bit value contained in the specified r. Stores the 32-bit result in R. Bit 32 of R contains the least significant bit of the result. The value of the CBIT is reset to 0. The value of LINK is indeterminate, and the condition codes are unchanged. This instruction cannot cause an overflow or generate an integer exception.

#### Note

MH r also has a register-to-register and an immediate form. See Appendix B for more information.

► N R,address  
 AND Fullword  
 0 0 0 0 1 1 DR\3 TM\2 SR\3 BR\2  
 [ DISPLACEMENT\16 ]

Calculates an effective address, EA. Logically ANDs the value contained in the specified R with the 32-bit value contained in the location specified by EA. Stores the result in the specified R. Leaves the values of CBIT, LINK, and the condition codes unchanged.

#### Note

This instruction also has a register-to-register and an immediate form. See Appendix B for more information.

► NFYB address  
 Notify to Beginning  
 0 0 0 0 0 0 1 0 1 0 0 0 1 0 0 1  
 AP\32

Notifies on semaphore at address specified in second and third halfwords of the instruction. Uses LIFO (last in, first out) queueing. Does not clear the currently active interrupt. The values of CBIT, LINK, and the condition codes are indeterminate. For more information, see Chapter 9 of the System Architecture Reference Guide.

#### Note

This is a restricted instruction.

► NFYE address  
 Notify to End  
 0 0 0 0 0 0 1 0 1 0 0 0 1 0 0 0  
 AP\32

Notifies on semaphore at the address specified in second and third halfwords of the instruction. Uses FIFO (first in, first out) queueing. Does not clear the currently active interrupt. The values of CBIT, LINK, and the condition codes are indeterminate. For more information, see Chapter 9 of the System Architecture Reference Guide.

#### Note

This is a restricted instruction.

► NH r,address  
 AND Halfword  
 0 0 1 0 1 1 DR\3 TM\2 SR\3 BR\2  
 [ DISPLACEMENT\16 ]

Calculates an effective address, EA. Logically ANDs the value contained in the specified r with the 16-bit value contained in the location specified by EA. Stores the result in r. Leaves the values of CBIT, LINK, and the condition codes unchanged.

Note

NH also has a register-to-register and an immediate form. See Appendix B for more information.

► NOP  
 No Operation  
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1

Does nothing. Leaves the values of CBIT, LINK, and the condition codes unchanged.

► O R, address  
 OR Fullword  
 0 1 0 0 1 1 DR\3 TM\2 SR\3 BR\2  
 [ DISPLACEMENT\16 ]

Calculates an effective address, EA. Logically ORs the value contained in the specified R with the 32-bit value contained in the location specified by EA. Stores the result in the specified R. Leaves the values of CBIT, LINK, and the condition codes unchanged.

#### Note

This instruction also has a register-to-register and an immediate form. See Appendix B for more information.

► OH r, address  
 OR Halfword  
 0 1 1 0 1 1 R\3 TM\2 SR\2 BR\2  
 [ DISPLACEMENT\16 ]

Calculates an effective address, EA. Logically ORs the value contained in the specified r with the 16-bit value contained in the location specified by EA. Stores the result in r. Leaves the values of CBIT, LINK, and the condition codes unchanged.

#### Note

This instruction also has a register-to-register and an immediate form. See Appendix B for more information.

► OTK r  
 Output Keys  
 0 1 1 0 0 0 R\3 0 1 1 1 0 0 1

Stores the contents of the specified r in the keys. Resets bits 15 to 16 of the keys to 0. Loads CBIT, LINK, and the condition codes from the specified r as a result of the operation. If this instruction is executed in Ring 0, it inhibits interrupts during execution of the next instruction.

► PCL address  
 Procedure Call  
 1 0 0 1 1 0 0 0 1 TM\2 SR\3 ER\2  
 DISPLACEMENT\16

See Chapter 8 of the System Architecture Reference Guide for a complete description of this instruction. Sets CBIT, LINK, and the condition codes to the values contained in the ECB.

#### Note

When arguments are to be transferred to the called procedure, this instruction uses GR5 and GR7, destroying the previous contents of these registers. XB is updated if an AP has the S bit = 0. The contents of GR5, GR7, and XB remain unchanged if no arguments are transferred. The contents of the condition codes, CBIT, and LINK are not correctly saved in the ECB along with the rest of the caller's keys.

► PID R  
 Position for Integer Divide  
 0 1 1 0 0 0 R\3 0 1 0 1 0 1 0

Positions a register for integer divide. Loads the contents of the specified R into R+1. Extends the sign of R (bit 1) into bits 2 to 32 of R. Leaves the values of CBIT, LINK, and the condition codes unchanged.

#### Note

R must be a even numbered register.

► PIDH r  
 Position r for Integer Divide  
 0 1 1 0 0 0 R\3 0 1 0 1 0 1 1

Moves the contents of the specified r (bits 1 to 16 of R) into bits 17 to 32 of R. Extends the contents of bit 1 of r into bits 2 to 16 of R. Leaves the values of CBIT, LINK, and the condition codes unchanged.

► PIM R  
 Position After Multiply  
 0 1 1 0 0 0 R\3 0 1 0 1 0 0 0

Checks bit 1 of R+1 to see if it is the same as all the bits in the specified R, and then moves the contents of R+1 into R. If bit 1 of R+1 was not the same as all the bits in R, an overflow occurs which causes an integer exception. If no integer exception occurs, CBIT is reset to 0. The values of LINK and the condition codes are indeterminate.

If an integer exception occurs and bit 8 in the keys contains 0, the PIM instruction sets CBIT to 1. If bit 8 contains 1, the instruction sets CBIT to 1 and causes an integer exception fault. For more information, see Chapter 10 of the System Architecture Reference Guide.

#### Note

R must be an even numbered register.

► PIMH r  
 Position r after Multiply  
 0 1 1 0 0 0 R\3 0 1 0 1 0 0 1

Checks the contents of bit 17 of the specified R to see if it has the same value as do all of bits 1 to 16 of R, and then moves the contents of bits 17 to 32 into bits 1 to 16. If bit 17 was different from all of bits 1 to 16, an integer exception occurs. If no integer exception occurs, CBIT is reset to 0. The values of LINK and the condition codes are indeterminate.

If an integer exception occurs and bit 8 of the keys contains 0, the instruction sets CBIT to 1. If bit 8 contains a 1, the instruction sets CBIT to 1 and causes an integer exception fault. See Chapter 10 of the System Architecture Reference Guide for more information.

#### Note

To position bits 17 to 32 of R in bits 1 to 16 of R, PIMH can modify all 32 bits of R, meaning that the contents of bits 17 to 32 of R are indeterminate at the end of this instruction.

► PRTN  
 Procedure Return  
 0 0 0 0 0 0 0 1 1 0 0 0 1 0 0 1

Deallocates the stack frame created for the executing procedure and returns to the environment of the procedure that called it.

To deallocate the frame, the instruction stores the current value of the stack base register into the free pointer. It then restores the caller's state by loading the caller's program counter, stack base register, linkage base register, and keys with the values contained in the frame being deallocated. Sets bits 15 to 16 of the keys to 0.

Loads the ring number in the program counter with the logical OR (weaker) of the saved program counter ring and the current ring number. This process prevents inward returns but also allows returns from gated calls to work properly.

► PTLB  
 Purge TLB  
 0 0 0 0 0 0 0 0 0 0 1 1 0 1 0 0

GR2 contains the address of a physical page, right justified. Based on the value of GR2 bit 1, PTLB purges either the first 128 locations of the STLB (i.e., not the IOTLB), or a specified physical page. If GR2 bit 1 contains a 1, the instruction performs a complete purge. If GR2 bit 1 contains a 0, the instruction purges the page specified by GR2. Leaves the values of CBIT, LINK, and the condition codes indeterminate. See Chapters 1, 4, and 11 of the System Architecture Reference Guide for more information about the STLB and IOTLB.

#### Note

This is a restricted instruction.

On the 750, 850, 2350 to 9955 II, insert a CRE (Clear E) instruction before PTLB. Since PTLB uses E (GR3 in I mode) as a pointer, the CRE zeros GR3 before PTLB manipulates it. If an interrupt occurs during PTLB's execution, GR3 points to the location PTLB is currently purging. PTLB leaves the contents of GR3 in an undefined state at the end of its execution.

► QFAD address  
 Quad Precision Floating Add  
 0 1 1 1 1 0 1 1 0 TM\2 SR\3 BR\2  
 DISPLACEMENT\16

Calculates an effective address, EA. Adds the 112-bit, quad precision number contained in the locations specified by EA to the contents of QAC. (See Chapter 6 of the System Architecture Reference Guide.) Normalizes the result, if necessary, and loads it into QAC. An overflow or underflow causes a floating-point exception. If no floating-point exception occurs, CBIT is reset to 0. The values of LINK and the condition codes are indeterminate.

If a floating-point exception occurs and bit 7 of the keys contains a 1, the instruction sets CBIT to 1. If bit 7 contains a 0, the instruction sets CBIT to 1 and causes a floating-point exception fault. See Chapter 10 of the System Architecture Reference Guide.

#### Note

If QFAD is used for any earlier system listed in "About This Book", an unimplemented instruction (UII) fault occurs. (See Chapter 10 of the System Architecture Reference Guide.)

► QFC address  
 Quad Precision Floating Compare  
 1 0 0 1 1 0 1 1 1 TM\2 SR\3 BR\2  
 DISPLACEMENT\16

Calculates an effective address, EA. Compares the contents of QAC (explained in Chapter 6 of the System Architecture Reference Guide) to the 112-bit contents of the location specified by EA. Leaves the values of CBIT and LINK unchanged. Sets the condition codes (CC) to the outcome of the comparison as shown below.

<u>Condition</u>	<u>CC</u>
Contents of QAC > contents of location specified by EA.	GT
Contents of QAC = contents of location specified by EA.	EQ
Contents of QAC < contents of location specified by EA.	LT

On some processors, QFC works correctly only on normalized numbers as follows. The comparison has a maximum of three sequential stages: first the signs, then the exponents, and finally the fractions of the two numbers are compared for equality. If the comparison during any one of these stages reveals an inequality, the results are returned and the instruction ends. Unnormalized numbers are unexpected and produce unexpected results. Other processors actually perform a subtract, resulting in a proper comparison.

Note

If QFC is used for any earlier system listed in "About This Book", an unimplemented instruction (UII) fault occurs. (See Chapter 10 of the System Architecture Reference Guide.)

**QFCM**

Quad Precision Floating Complement

1 1 0 0 0 0 0 1 0 1 1 1 1 0 0 0

Forms the two's complement of the value contained in QAC. (See Chapter 6 of the System Architecture Reference Guide.) Normalizes the result, if necessary, and stores it in QAC. An underflow or overflow causes a floating-point exception. If no floating-point exception occurs, CBIT is reset to 0. The values of LINK and the condition codes are indeterminate.

If a floating-point exception occurs and bit 7 of the keys contains a 1, the instruction sets CBIT to 1. If bit 7 contains a 0, the instruction sets CBIT to 1 and causes a floating-point exception fault. See Chapter 10 of the System Architecture Reference Guide.

Note

If QFCM is used for any earlier system listed in "About This Book", an unimplemented instruction (UII) fault occurs. (See Chapter 10 of the System Architecture Reference Guide.)

**QFDV address**

Quad Precision Floating Point Divide

1 0 0 1 1 0 1 1 0 TM\2 SR\3 ER\2

DISPLACEMENT\16

Calculates an effective address, EA. Divides the contents of QAC by the 112-bit contents of the location specified by EA. (See Chapter 6 of the System Architecture Reference Guide.) Normalizes the result, if necessary, and stores the whole quotient into QAC. An overflow, underflow, or divide by 0 causes a floating-point exception. If no floating-point exception occurs, CBIT is reset to 0. The values of LINK and the condition codes are indeterminate.

If a floating-point exception occurs and bit 7 of the keys contains a 1, the instruction sets CBIT to 1. If bit 7 contains a 0, the instruction sets CBIT to 1 and causes a floating-point exception fault. See Chapter 10 of the System Architecture Reference Guide.

Note

If QFDV is used for any earlier system listed in "About This Book", an unimplemented instruction (UII) fault occurs. (See Chapter 10 of the System Architecture Reference Guide.)

► QFLD address  
 Quad Precision Floating Load  
 0 1 1 1 1 0 1 0 0 TM\2 SR\3 BR\2  
 DISPLACEMENT\16

Calculates an extended, augmented effective address, EA. Performs one of the following actions with the value contained in the location specified by EA. Loads bits 1 to 112 into QAC and zeros QAC bits 113 to 128, or loads 128 bits into QAC. In either case, there is no normalization of the result. (See Chapter 6 of the System Architecture Reference Guide for more information.) Leaves the values of CBIT, LINK, and the condition codes unchanged.

Note

If QFLD is used for any earlier system listed in "About This Book", an unimplemented instruction (UII) fault occurs. (See Chapter 10 of the System Architecture Reference Guide.)

► QFMP address  
 Quad Precision Floating Point Multiply  
 1 0 0 1 1 0 1 0 1 TM\2 SR\3 BR\2  
 DISPLACEMENT\16

Calculates an effective address, EA. Multiplies the contents of QAC by the 112-bit contents of the location specified by EA. (See Chapter 6 of the System Architecture Reference Guide.) Normalizes the result, if necessary, and stores it into QAC. An overflow or underflow causes a floating-point exception. If no floating-point exception occurs, CBIT is reset to 0. The values of LINK and the condition codes are indeterminate.

If a floating-point exception occurs and bit 7 of the keys contains a 1, the instruction sets CBIT to 1. If bit 7 contains a 0, the instruction sets CBIT to 1 and causes a floating-point exception fault. See Chapter 10 of the System Architecture Reference Guide for more information.

Note

If QFMP is used for any earlier system listed in "About This Book", an unimplemented instruction (UII) fault occurs. (See Chapter 10 of the System Architecture Reference Guide.)

► QFSB address  
 Quad Precision Floating Point Subtract  
 0 1 1 1 1 0 1 1 1 TM\2 SR\3 BR\2  
 DISPLACEMENT\16

Calculates an effective address, EA. Subtracts the 112-bit contents of the locations specified by EA from the contents of QAC. (See Chapter 6 of the System Architecture Reference Guide.) Normalizes the result, if necessary, and loads it into QAC. An overflow or underflow causes a floating-point exception. If no floating-point exception occurs, CBIT is reset to 0. The values of LINK and the condition codes are indeterminate.

If a floating-point exception occurs and bit 7 of the keys contains a 1, the instruction sets CBIT to 1. If bit 7 contains a 0, the instruction sets CBIT to 1 and causes a floating-point exception fault. See Chapter 10 of the System Architecture Reference Guide for more information.

Note

If QFSB is used for any earlier system listed in "About This Book", an unimplemented instruction (UII) fault occurs. (See Chapter 10 of the System Architecture Reference Guide.)

► QFST address  
 Quad Precision Floating Store  
 0 1 1 1 1 0 1 0 1 TM\2 SR\3 BR\2  
 DISPLACEMENT\16

Calculates an effective address, EA. Stores the contents of QAC into the 128 bits of memory specified by EA. Leaves the values of LINK, CBIT, and the condition codes unchanged.

Note

QFST does not normalize the result before storing it into the specified memory location.

If QFST is used for any earlier system listed in "About This Book", an unimplemented instruction (UII) fault occurs. (See Chapter 10 of the System Architecture Reference Guide.)

► QINQ  
 Quad to Integer, in Quad Convert  
 1 1 0 0 0 0 0 1 0 1 1 1 1 0 1 0

Strips the fractional portion of QAC as described in Table 3-5.

Table 3-5  
 QINQ Actions

Exponent Value	Action
'337 <= Exp	No operation.
'200 < Exp < '337	If sign >= 0, strip fractional part of QAC for result. If sign < 0 and fractional part <> 0, strip fractional part of QAC and increment integer portion of QAC by 1. If sign < 0 and fractional part = 0, no action is done.
'200 = Exp	If sign >= 0, result = 0. If sign < 0 and bits 2 to 96 = 0 result = -1. If sign < 0 and bits 2 to 96 <> 0 result = 0.
'200 > Exp	Result = 0.

QINQ can cause a floating-point exception. This exception does not alter the contents of QAC. If no exception occurs, the instruction resets CBIT to 0. The values of LINK and the condition codes are indeterminate.

If a floating-point exception occurs and bit 7 of the keys contains a 1, the instruction sets CBIT to 1. If bit 7 contains a 0, the instruction sets CBIT to 1 and causes a floating-point exception fault. See Chapter 10 of the System Architecture Reference Guide for more information.

#### Note

If QINQ is used for any earlier system listed in "About This Book", an unimplemented instruction (UII) fault occurs. (See Chapter 10 of the System Architecture Reference Guide.)

► **QIQR**  
 Quad to Integer, in Quad Convert Rounded  
 1 1 0 0 0 0 0 1 0 1 1 1 1 0 1 1

Strips the fractional portion of QAC as described in Table 3-6.

Table 3-6  
 QIQR Actions

Exponent Value	Action
'337 <= Exp	No operation.
'177 < Exp < '337	If sign >= 0, round.* If sign < 0 and fractional part <> 0.5,** round and strip the fractional part of QAC.
Exp = '177	If sign >= 0, result = 0. If sign < 0 and bits 2 to 96 = 0, result = -1. If sign < 0 and bits 2 to 96 <> 0, result = 0. For all cases increment integer part by 1 if it exists and the most significant bit of QAC = 1.
Exp < '177	The result is 0.

\* Rounding occurs if the MSB of the QAC fraction is 1. For example, add the MSB of the QAC fraction to itself and carry out to the QAC integer.

\*\* 0.5 implies a QAC fraction with the MSB = 1 and all other bits = 0.

QIQR can cause a floating-point exception. This exception does not alter the contents of QAC. If no exception occurs, the instruction sets CBIT to 0. The values of LINK and the condition codes are indeterminate.

If a floating-point exception occurs and bit 7 of the keys contains a 1, the QIQR instruction sets CBIT to 1. If bit 7 contains a 0, the instruction sets CBIT to 1 and causes a floating-point exception fault. See Chapter 10 of the System Architecture Reference Guide for more information.

Note

If QIQR is used for any earlier system listed in "About This Book", an unimplemented instruction (UII) fault occurs. (See Chapter 10 of the System Architecture Reference Guide.)

## INSTRUCTION SETS GUIDE

► **RBQ** r,address  
Remove Entry From Bottom of Queue  
0 1 1 0 0 0 R\3 1 0 1 1 0 1 1  
AP\32

The address pointer in this instruction points to the QCB for a queue. The instruction removes the entry from the bottom of the referenced queue and loads it into the specified r. If the queue was not empty, this instruction sets the condition codes to reflect not equal to. If the queue was empty, resets r to 0 and sets the condition codes to reflect equal to. Leaves the values of CBIT and LINK unchanged.

► **RCB**  
Reset CBIT to 0  
1 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0

Resets CBIT to 0. Leaves the values of LINK and the condition codes unchanged.

► **RMC**  
Reset Machine Check Flag to 0  
0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1

Resets the machine check mode (bits 15 to 16 of the modals) to 0. Leaves the values of CBIT, LINK, and the condition codes unchanged. Inhibits interrupts for the next instruction.

### Note

This is a restricted instruction.

► **ROT** R,address  
Rotating Shift  
0 1 0 1 0 0 DR\3 TM\2 SR\3 BR\2  
DISPLACEMENT\16

Calculates an effective address, EA. Interprets bits 17 to 32 of EA as a shift command, as shown in Table 3-7.

Table 3-7  
EA Format for ROT Shift Command

Bits	Value	Interpretation
17	0	Shift left.
	1	Shift right.
18	0	Word shift (32 bits).
	1	Halfword shift (16 bits).
19 to 26	---	Ignored.
27 to 32	---	Values specify the two's complement of the number of bits to shift. A value of 0 indicates a shift of 64 places; of -1, 1 place; of -63, 63 places; and so on.

Uses EA to perform a rotating shift on the contents of the specified R. Stores the shifted result in R. CBIT and LINK contain the value of the last bit shifted out. Leaves the values of the condition codes unchanged.

► RREST address  
Restore Registers  
0 0 0 0 0 0 0 1 1 1 0 0 1 1 1 1  
AP\32

Calculates an effective address, EA, from the 32-bit address pointer in the instruction. This specifies the starting address of a save area for the general, floating, and XB registers. Restores the contents of these registers from this save area.

The save area format is shown in Table 3-8. Bits 1 to 16 of the save area are a save mask, whose format appears in Figure 3-3. A mask bit value of 1 means that the corresponding register had nonzero contents that have been saved in the save area; a mask bit value of 0 means that the corresponding register's contents were 0. Leaves the values of CBIT, LINK, and the condition codes unchanged.

Table 3-8  
RRST and R SAV Save Area Format

Offset #	Contents
1	Save mask
2 to 5	FR1 (F)
6 to 9	FRO
10 to 11	X, GR7
12 to 13	GR6
14 to 15	Y, S, GR5
16 to 17	GR4
18 to 19	E, GR3
20 to 21	A, B, L, GR2
22 to 23	GR1
24 to 25	GRO
26 to 27	XB

1	4	5	6	7	8	9	10	11	12	13	14	15	16
0000	FR1	FRO	GR7	GR6	GR5	GR4	GR3	GR2	GR1	GRO			

Save Mask Format, RRST and R SAV Instructions  
Figure 3-3

► R SAV address  
Save Registers  
0 0 0 0 0 0 0 1 1 1 0 0 1 1 0 1  
AP\32

Calculates an effective address, EA, from the 32-bit address pointer in the instruction. This specifies the starting address of a save area for the general, floating, and XB registers. Saves the nonzero contents of these registers in the save area.

The save area format is shown in Table 3-8. Bits 1 to 16 of the save area are a save mask, whose format appears in Figure 3-3. This instruction sets the mask bit of each register as follows: to 1 if the register's contents have a nonzero value; to 0 if a 0 value. Leaves the values of CBIT, LINK, and the condition codes unchanged.

► RTQ r,address  
 Remove Entry From Top of Queue  
 0 1 1 0 0 0 R\3 1 0 1 1 0 1 0  
 AP\32

The address pointer in this instruction is to the QCB for a queue. The instruction removes the entry from the top of the referenced queue, and loads it into the specified r. If the queue was not empty, the instruction sets the condition codes to reflect not equal to 0. If the queue was empty, resets r to 0 and sets the condition codes to reflect equal to. Leaves the values of CBIT and LINK unchanged.

► RTS  
 Reset Time Slice  
 0 0 0 0 0 0 0 1 0 1 0 0 1 0 0 1

Valid for the 550-II, 750, 850, I450, and new processors.

GR2H (bits 1 to 16) contain a negative value that represents the number of milliseconds in the new time slice. The time slice is determined by counting ITH up every 1.024 milliseconds until zero when the time slice ends. Therefore, ITH is the two's complement of the number of milliseconds remaining in the time slice. The elapsed timer contains the total number of 1.024 millisecond units that have elapsed since process creation plus the full count of the current time slice. Combining ITH and ET by addition gives the total elapsed time.

RTS adds the current value of the interval timer (locations 16 to 17 of the PCB) to the contents of the elapsed timer (locations 10 to 11 of the PCB), then subtracts the contents of GR2H from the sum of the timers. Stores the result in the elapsed timer. Loads the contents of GR2H into the interval timer. Leaves the contents of GR2H unchanged. The values of CBIT, LINK, and the condition codes are unchanged. The addition performed by this instruction is equivalent to the following series of instructions:

```
LH    0,ITH /* Load GRO with contents of ITH.
SH    0,2  /* Subtract reset value in GR2H from GRO contents
PIDH  0    /* Sign extend the contents of GROH into bits
          /*      17 to 32 of GRO.
SRC    /* Skip next 16-bit halfword if CBIT is 0.
CMH    0    /* Complement GRO.
A      0,ET /* Add ITH and ET.
ST      0,ET /* Store result in ET.
STH    2,ITH /* Store GR2 contents in ITH.
```

#### Note

RTS is a restricted instruction.

► S R,address  
 Subtract Fullword  
 0 1 0 0 1 0 DR\3 TM\2 SR\3 BR\2  
 [ DISPLACEMENT\16 ]

Calculates an effective address, EA. Subtracts the 32-bit value contained in the location specified by EA from the value contained in the specified R. Stores the result in the specified R. If overflow occurs, an integer exception results. If no integer exception occurs, CBIT is reset to 0. LINK contains the borrow bit. The condition codes reflect the result of the operation. (See Appendix A.)

If an integer exception occurs and bit 8 of the keys contains 0, the instruction sets CBIT to 1. If bit 8 contains a 1, the instruction sets CBIT to 1 and causes an integer exception fault. See Chapter 10 of the System Architecture Reference Guide for more information.

#### Note

This instruction also has a register-to-register and an immediate form. See Appendix B for more information.

► SCB  
 Set CBIT to 1  
 1 1 0 0 0 0 0 1 1 0 0 0 0 0 0 0

Sets the value of CBIT to 1. The value of LINK is indeterminate. Leaves the values of the condition codes unchanged.

► SOC r,address  
 Store C Character  
 1 0 1 1 0 1 DR\3 TM\2 SR\3 BR\2  
 DISPLACEMENT\16

Uses the C language pointer to store a single character from bits 9 to 16 of the specified r into a location in memory. (Bits 1 to 8 of r are not modified and do not affect this operation.) When bit 4 of the C pointer contains 0, the character is stored into bits 1 to 8 of the address; if bit 4 of the pointer contains 1, the character is stored into bits 9 to 16 of the address. Leaves the values of the CBIT, LINK, and condition codes unchanged.

#### Note

The SOC instruction is valid only for general register relative and indirect forms of address formation. Other forms of address formation (including indexing) do not reliably generate the C language pointer.

In particular, do not use the immediate or register-to-register form with the SOC instruction because it would be interpreted as an ACP instruction. (SOC and ACP share the same opcode, but ACP uses the immediate and register-to-register form.) However, direct addressing will obtain the first byte (of two) pointed to by the effective address. This assumes that the base register used was loaded with a conventional 32-bit IP with the E bit reset.

If SOC is used for any earlier system listed in "About This Book", an unimplemented instruction (UII) fault occurs. (See Chapter 10 of the System Architecture Reference Guide.)

► SH r,address  
Subtract Halfword  
0 1 1 0 1 0 DR\3 TM\2 SR\3 BR\2  
[ DISPLACEMENT\16 ]

Calculates an effective address, EA. Subtracts the 16-bit value contained in the location specified by EA from the value contained in the specified r and stores the result in r. An overflow causes an integer exception. If no integer exception occurs, CBIT is reset to 0. LINK contains the borrow bit. The condition codes reflect the result of the operation. (See Appendix A.)

If an integer exception occurs and bit 8 of the keys contains 0, the instruction sets CBIT to 1. If bit 8 contains a 1, the instruction sets CBIT to 1 and causes an integer exception fault. See Chapter 10 of the System Architecture Reference Guide for more information.

#### Note

The SH instruction also has a register-to-register and an immediate form. See Appendix B for more information.

► SHA R,address  
Arithmetic Shift  
0 0 1 1 0 1 DR\3 TM\2 SR\3 BR\2  
DISPLACEMENT\16

Calculates an effective address, EA. Interprets bits 17 to 32 of EA as a shift command, as shown in Table 3-9.

Table 3-9  
EA Format for SHA Shift Command

Bits	Value	Interpretation
17	0	Shift left.
	1	Shift right.
18	0	Word shift (32 bits).
	1	Halfword shift (16 bits).
19 to 26	---	Ignored.
27 to 32	---	Values specify the two's complement of the number of bits to shift. A value of 0 indicates a shift of 64 places; of -1, 1 place; of -63, 63 places; and so on.

Uses EA to perform an arithmetic shift on the contents of the specified R, and stores the result of the shift in R.

For a right shift, CBIT and LINK contain the value of the last bit shifted out. The values of all other shifted-out bits are lost.

For a left shift, an overflow causes an integer exception. If there is no integer exception, CBIT is reset to 0. The value of LINK is indeterminate.

All shifts leave the values of the condition codes unchanged.

If an integer exception occurs and bit 8 of the keys contains 0, the instruction sets CBIT to 1. If bit 8 contains a 1, the instruction sets CBIT to 1 and causes an integer exception fault. See Chapter 10 of the System Architecture Reference Guide for more information.

► SHL R,address  
Logical Shift  
0 0 0 1 0 1 DR\3 TM\2 SR\3 BR\2  
DISPLACEMENT\16

Calculates an effective address, EA. Interprets bits 17 to 32 of EA as a shift command, as shown in Table 3-10.

Table 3-10  
EA Format for SHL Shift Command

Bits	Value	Interpretation
17	0	Shift left.
	1	Shift right.
18	0	Word shift (32 bits).
	1	Halfword shift (16 bits).
19 to 26	---	Ignored.
27 to 32	---	Values specify the two's complement of the number of bits to shift. A value of 0 indicates a shift of 64 places; of -1, 1 place; of -63, 63 places; and so on.

Uses EA to perform a logical shift on the contents of the specified R. Stores the shifted result in R. CBIT and LINK contain the value of the last bit shifted out. The values of all other shifted-out bits are lost. Leaves the values of the condition codes unchanged.

► SHL1 r  
Shift r Left 1  
0 1 1 0 0 0 R\3 0 1 1 1 1 1 0

Shifts the contents of the specified r to the left one bit and stores the result in r. CBIT and LINK contain the value of the bit shifted out. Leaves the values of the condition codes unchanged.

► SHL2 r  
Shift r Left 2  
0 1 1 0 0 0 R\3 0 1 1 1 1 1 1

Shifts the contents of the specified r to the left two bits and stores the result in r. CBIT and LINK contain the value of the last bit shifted out. The value of the first bit shifted out is lost. Leaves the values of the condition codes unchanged.

► **SHR1 r**  
 Shift r Right 1  
 0 1 1 0 0 0 R\3 1 0 1 0 0 0 0

Shifts the contents of the specified r to the right one bit and stores the result in r. CBIT and LINK contain the value of the bit shifted out. Leaves the values of the condition codes unchanged.

► **SHR2 r**  
 Shift r Right 2  
 0 1 1 0 0 0 R\3 1 0 1 0 0 0 1

Shifts the contents of the specified r to the right two bits and stores the result in r. CBIT and LINK contain the value of the last bit shifted out. The value of the first bit shifted out is lost. Leaves the values of the condition codes unchanged.

► **SL1 R**  
 Shift Register Left 1  
 0 1 1 0 0 0 R\3 0 1 1 1 0 1 0

Shifts the contents of the specified R to the left one bit and stores the result in R. CBIT and LINK contain the value of the bit shifted out. Leaves the values of the condition codes unchanged.

► **SL2 R**  
 Shift Register Left 2  
 0 1 1 0 0 0 R\3 0 1 1 1 0 1 1

Shifts the contents of the specified R to the left two bits and stores the result in R. CBIT and LINK contain the value of the last bit shifted out; the value of the first bit shifted out is lost. Leaves the values of the condition codes unchanged.

► **SR1 R**  
 Shift Register Right 1  
 0 1 1 0 0 0 R\3 0 1 1 1 1 0 0

Shifts the contents of the specified R to the right one bit and stores the result in R. CBIT and LINK contain the value of the bit shifted out. Leaves the values of the condition codes unchanged.

► SR2 R  
Shift Register Right 2  
0 1 1 0 0 0 R\3 0 1 1 1 1 0 1

Shifts the contents of the specified R to the right two bits and stores the result in R. CBIT and LINK contain the value of the last bit shifted out; the value of the first bit shifted out is lost. Leaves the values of the condition codes unchanged.

► SSM R  
Set Sign Minus  
0 1 1 0 0 0 R\3 0 1 0 0 0 1 0

Sets bit 1 of the specified R to 1. Leaves the values of CBIT, LINK, and the condition codes unchanged.

► SSP R  
Set Sign Plus  
0 1 1 0 0 0 R\3 0 1 0 0 0 1 1

Resets bit 1 of the specified R to 0. Leaves the values of CBIT, LINK, and the condition codes unchanged.

► SSSN  
Store System Serial Number  
0 1 0 0 0 0 0 0 1 1 0 0 1 0 0 0

This instruction is applicable only for the 2350 to the 9955 II. A 14-character system identifier programmed into these processors during manufacturing consists of a 2-character plant location code followed by a 12-digit number. (These characters and numbers are in 7-bit ASCII format.) SSSN writes this system identifier into a 16-halfword block at the address specified by the XB register. (A halfword is 16 bits.) The first 8 halfwords of this block hold the system serial number string as provided by manufacturing; the remaining halfwords are reserved for future expansion and are 0.

Leaves the values of CBIT, LINK, and the condition codes indeterminate.

#### Note

If SSSN is used for any earlier system listed in "About This Book", an unimplemented instruction (UII) fault occurs. (See Chapter 10 of the System Architecture Reference Guide.)

► ST R,address  
 Store Fullword  
 0 1 0 0 0 1 DR\3 TM\2 SR\3 BR\2  
 DISPLACEMENT\16

Calculates an effective address, EA. Stores the contents of the specified R into the location specified by EA. Leaves the values of the CBIT, LINK, and condition codes unchanged.

► STAR R,address  
 Store Addressed Register  
 1 0 1 1 0 0 DR\3 TM\2 SR\3 BR\2  
 DISPLACEMENT\16

Calculates a 32-bit (word) effective address, EA. Stores the contents of the specified R into the register location specified by the offset portion of EA. Bit 2 and bit 12 of the offset portion of EA determine the actions of this instruction, as shown in Table 3-11.

Table 3-11  
 STAR Actions

Bit 2	Bit 12	Action
1*	----	Ignore bits 1 and 3 to 9. The offset portion of EA specifies an absolute register number from 0 to '377.
0*	1	Bits 13 to 16 of the offset portion of EA specify one of the registers '20 to '37 in the current register set.
0	0	Bits 13 to 16 of the offset portion of EA specify one of the registers 0 to '17 in the current register set.

\*This is a restricted instruction.

Leaves the values of CBIT and LINK unchanged. The values of the condition codes are indeterminate. See Chapter 9 of the System Architecture Reference Guide for more information about register sets.

Note

Do not use the STAR instruction to write into the procedure base, keys, or modals. You can use LPSW to change any of these three registers. In addition, you can use a control transfer to change the procedure base, or a mode control operation to change the keys or modals. Under no circumstances should you try to change the value of the current register set bits contained in the modals.

If the current ring is not 0 and EA is outside the range of 0 to '17 inclusive, any access causes an RXM violation.

► STC flr,r  
Store Character  
0 1 1 0 0 0 R\3 1 1 1 FLR 1 1 0

If the contents of the specified FLR are nonzero, the instruction stores the contents of bits 9 to 16 of the specified r into the character byte address contained in the associated FAR. Updates the contents of the appropriate FAR so that they point to the next character. Decrements the contents of the specified FLR by 1. Sets the condition codes to NE.

If the contents of the specified FLR are 0, the instruction sets the condition codes to EQ and does not store a character.

The instruction leaves the values of LINK and CBIT unchanged.

Note

When the instruction specifies FLR0, FAR0 is used. When the instruction specifies FLR1, FAR1 is used.

► STCD R,address  
Store Conditional Fullword  
0 1 1 0 0 0 R\3 1 0 1 1 1 1 1  
AP\32

Compares the contents of R+1 and the contents of the 32-bit location referenced by the specified address pointer. If the two values are equal, the instruction stores the contents of R in that referenced location. If the two values are not equal, execution continues with the next instruction. STCD is an interlocked operation, guaranteed to work in a multiprocessor.

Leaves the values of CBIT and LINK unchanged. The condition codes indicate reflect the result of the comparison. (See Appendix A.)

Note

R must be an even numbered register.

► **STCH r,address**  
 Store Conditional Halfword  
 0 1 1 0 0 0 R\3 1 0 1 1 1 0  
 AP\32

Compares the contents of bits 17 to 32 of the specified R with the contents of the location referenced by the specified address pointer. If the two values are equal, the instruction stores the contents of r into that referenced location. If the two values are not equal, execution continues with the next instruction. Leaves the values of CBIT and LINK unchanged. Sets the condition codes to EQ if the store occurs and to NE if not.

The comparison and store will not be separated by execution of other instructions. Therefore, no instruction can alter the contents of the specified memory location between the compare and the store.

Note

This instruction is useful when two cooperating, sequential processes are manipulating shared data. It is interlocked against direct memory I/O. This means you can use it to interlock a process with a DMA, DMC, or DMQ channel, as well as to interlock a memory location that is possibly accessed by I/O.

► **STEX R**  
 Stack Extend  
 0 1 1 0 0 0 R\3 0 0 1 0 1 1 1

Extends the length of the procedure stack. The designated R contains a 32-bit number that specifies the halfword size of the extension. (A halfword is 16 bits.)

The firmware rounds up the number contained in the specified R to an even number of halfwords. The instruction uses this value to allocate a block of memory to the procedure stack. The extension and the initial stack segment do not have to be contiguous, since there may not have been enough room left in the initial stack to contain a complete frame.

Returns a segment number/offset number in the specified R that specifies the starting address of the extension. The extension is automatically deallocated when the current procedure completes execution. There is no limit on the number of extensions you can make.

A stack fault occurs if there is no room for the extension. The values of CBIT, LINK, and the condition codes are indeterminate. See Chapters 8 and 10 of the System Architecture Reference Guide for more information about this instruction, stacks, and stack faults.

► STFA far,address  
 Store FAR  
 0 0 0 0 0 0 1 0 1 1 0 1 FAR 0 0 0  
 AP\32

Stores the specified FAR contents as a hardware recognizable indirect pointer at the memory location referenced by the specified address pointer. If the bit number field of the specified FAR contains 0, the instruction stores the first 32 bits (two halfwords) of the pointer and clears the pointer's extend bit to 0. If the bit number field of the specified FAR does not contain 0, the instruction saves all 48 bits (three halfwords) of the pointer and sets the pointer's extend bit to 1. Leaves the values of CBIT, LINK, and the condition codes indeterminate.

► STH r,address  
 Store Halfword  
 0 1 1 0 0 1 DR\3 TM\2 SR\3 BR\2  
 DISPLACEMENT\16

Calculates an effective address, EA. Stores the contents of the specified r into the 16-bit location specified by EA. Leaves the values of CBIT, LINK, and the condition codes unchanged.

► STPM  
 Store Processor Model Number  
 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0

Stores the CPU model number and microcode revision number in an 8-halfword field. (A halfword is 16 bits.) XB contains a pointer to the field in memory. Table 3-12 shows the format of the field.

Table 3-12  
STPM Memory Field Format

Halfword	Name	Description
1 to 2	Processor Model Number	Contains a code specifying the machine: 0L - 400/500, no Rev B microcode 1L - 400, Rev. B microcode 2L - Reserved 3L - 350 4L - 450/550 5L - 750 6L - 650 7L - 250 8L - 850 9L - 250-II 10L - 550-II 11L - 2250 15L - 9950 16L - 9650 17L - 2550 18L - 9955 19L - 9750 21L - 2350 22L - 2655 23L - 9655 25L - 2450 30L - 9955 II 31L - 2755 34L - 6350 42L - 9755
3 to 4	Microcode Revision	Offset 3: Bits 1 to 8 Reserved Bits 9 to 16 Manufacturing microcode revision number Offset 4: Bits 1 to 16 Engineering microcode revision number
5	Processor Line	Specifies options enabled for this machine: Bits 1 to 15 Reserved; must be 0 Bit 16 Marketing segment specification bit
6	Extended Microcode ID	To be implemented.
7 to 8	---	Reserved for future use.

The STPM instruction leaves the values of CBIT, LINK, and the condition codes unchanged.

Note

STPM is a restricted instruction.

► STTM  
 Store Process Timer  
 0 0 0 0 0 0 0 1 0 1 0 0 1 0 0 0

Valid for the 550-II, 850, I450, and 2350 to 9955 II.

The current process time is represented by the sum of the 32-bit elapsed time (stored in the PCB) and the 32-bit interval timer (contained in the CPU hardware). Bit 17 of the elapsed time is equivalent in weight to bit 1 of the interval time. This operation is equivalent to the following sequence of instructions. (Register 0 is not actually modified by the STTM instruction.)

```
LDAR    0, PB% + '25    /* Get PCB address.
A        0, = '10L      /* Offset of elapsed time.
ST       0, TEMP1       /* Elapsed time address -> Temp.
LDAR    0, PB% + '30    /* Read timer.
IRH      0              /* Store low order
STH      0, XB% + 2     /* 16 bits.
IRH      0              /* Adjust
PIDH     0              /* weighting.
A        0, TEMP1, *    /* Add elapsed time.
ST       0, XB% + 0
```

Leaves the values of the CBIT, LINK, and condition codes indeterminate. This instruction is not implemented on the 2250.

► SVC  
 Supervisor Call  
 0 0 0 0 0 0 0 1 0 1 0 0 0 1 0 1

Supervisor call. Generates a directed fault. Leaves the values of CBIT, LINK, and the condition codes unchanged.

This instruction allows you to make an operating system request that is addressing mode independent. By software convention, this instruction sends an operation code and pointers to the operating system to generate a fault. For more information, refer to Chapter 10 of the System Architecture Reference Guide.

► TC R  
Two's Complement Register  
0 1 1 0 0 0 R\3 0 1 0 0 1 1 0

Forms the two's complement of the contents of the specified R and stores the result in R. An overflow causes an integer exception. If there is no integer exception, CBIT is reset to 0. The value of LINK is indeterminate. The condition codes reflect the result of the operation. (See Appendix A.)

If an integer exception occurs and bit 8 of the keys contains 0, the instruction sets CBIT to 1. If bit 8 contains a 1, the instruction sets CBIT to 1 and causes an integer exception fault. See Chapter 10 of the System Architecture Reference Guide for more information.

► TCH r  
Two's Complement r  
0 1 1 0 0 0 R\3 0 1 0 0 1 1 1

Forms the two's complement of the contents of the specified r and stores the result in r. An overflow causes an integer exception. If there is no integer exception, CBIT is reset to 0. The value of LINK is indeterminate. The condition codes reflect the result of the operation. (See Appendix A.)

If an integer exception occurs and bit 8 of the keys contains 0, the instruction sets CBIT to 1. If bit 8 contains a 1, the instruction sets CBIT to 1 and causes an integer exception fault. See Chapter 10 of the System Architecture Reference Guide for more information.

► TCNP address  
Test C Null Pointer  
1 1 1 1 1 0 1 1 0 TM\2 SR\3 BR\2  
[ DISPLACEMENT\16 ]

Calculates an effective address, EA. Tests bits 4 to 32 of the C language pointer in the location specified by EA for zero. When these bits are zero, this instruction sets the condition codes equal to zero; otherwise the condition codes are set not equal to zero. The values of the CBIT and LINK are unchanged.

#### Note

The TCNP instruction also has a register addressing form. The syntax and format for this form of TCNP is:

TCNP R  
0 1 1 0 0 0 R\3 1 1 1 1 0 0 0

(The expected form for TCNP register addressing would be

1 1 1 1 1 0 1 1 0 0 0 SR\3 0 0

but this is, in fact, unimplemented.)

If TCNP is used for any earlier system listed in "About This Book", an unimplemented instruction (UII) fault occurs. (See Chapter 10 of the System Architecture Reference Guide.)

► TFLR flr,R  
Transfer FLR to Register  
0 1 1 0 0 0 R\3 1 1 1 FLR 0 1 1

Transfers the contents of the specified FLR into the specified R. Leaves the values of CBIT, LINK, and the condition codes unchanged.

► TM address  
Test Memory Fullword  
1 0 0 1 1 0 1 0 0 TM\2 SR\3 BR\2  
DISPLACEMENT\16

Calculates an effective address, EA. Sets the condition codes according to the numerical value of the 32-bit contents of the location specified by EA. (See Appendix A.) Leaves the values of LINK and CBIT unchanged.

► TMH address  
Test Memory Halfword  
1 0 1 1 1 0 1 0 0 TM\2 SR\3 BR\2  
DISPLACEMENT\16

Calculates an effective address, EA. Sets the condition codes according to the numerical value of the contents of bits 1 to 16 of the location specified by EA. (See Appendix A.) Leaves the values of LINK and CBIT unchanged.

► TRFL flr,R  
Transfer Register to FLR  
0 1 1 0 0 0 R\3 1 1 1 FLR 1 0 1

Transfers the contents of R into the specified FLR. Clears bits 1 to 11 of R to 0 so that bits 1 to 6 of the specified FLR will be 0. Leaves the values of CBIT, LINK, and the condition codes unchanged.

Note

The TRFL instruction allows you to load the specified FLR with a value computed at execution time. The maximum allowable integer you can load is  $2^{20}$ . This number is 21 bits wide and equals the number of bits in a 64K segment.

► TSTQ r, address  
 Test Queue  
 0 1 1 0 0 0 R\3 1 0 0 0 1 0 0  
 AP\32

The address pointer in this instruction points to the QCB of a queue. This instruction tests the referenced queue and sets r to equal the number of items in the queue. Sets the condition codes to EQ when the queue is empty. If the queue is not empty, the instruction sets the condition codes to NE. Leaves the values of CBIT and LINK unchanged.

► WAIT address  
 Wait  
 0 0 0 0 0 0 0 0 1 1 0 0 1 1 0 1  
 AP\32

The address pointer in this instruction points to a 16-bit semaphore counter, C. The instruction increments C. If C is greater than 0, either the resource is not available, or the event has not occurred. Removes the PCB from the ready list, suspending the process, and adds it to the wait list associated with the semaphore. It then makes the register set available, turns off the process timer, and goes to the dispatcher to find another process to run. The dispatcher enables interrupts.

If C is less than or equal to 0, the currently executing process continues.

If the instruction places the PCB on the wait list, no general registers are saved. This means that a process cannot depend on these registers to be intact after this instruction occurs. This instruction potentially clears the general, floating, and XB registers.

Leaves CBIT, LINK, and the condition codes unchanged.

For more information about semaphores, the dispatcher, PCBs, and wait lists, refer to Chapter 9 of the System Architecture Reference Guide.

#### Note

This is a restricted instruction.

► X R,address  
 Exclusive OR Fullword  
 1 0 0 0 1 1 DR\3 TM\2 SR\3 BR\2  
 [ DISPLACEMENT\16 ]

Calculates an effective address, EA. Performs an exclusive OR of the contents of the specified R with the 32-bit value contained in the location specified by EA. Stores the result in the specified R. Leaves the values of CBIT, LINK, and the condition codes unchanged.

#### Note

This instruction also has a register-to-register and an immediate form. See Appendix B for more information.

► XAD  
 Decimal Add  
 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0

Performs a decimal arithmetic operation under control of FAR0, FAR1, and GR2.

FAR0 contains the address of field 1. FAR1 contains the address of field 2. GR2 contains the control word; fields B and C of the control word specify the decimal operation to be performed, as shown in Table 3-13.

Table 3-13  
 XAD Decimal Operations

B	CB	Operation	Destination
0	0	+F1+F2	F2
0	1	+F1-F2	F2
1	0	-F1+F2	F2
1	1	-F1-F2	F2

The scale differential field in the control word specifies the difference in the decimal point alignment between F1 and F2, as follows:

<u>SD</u>	<u>Relation of F1 and F2</u>
SD>0	F1 > F2
SD=0	F1 = F2
SD<0	F1 < F2

If the T bit is set to 1, the results are forced positive. If the add operation results in an overflow, a decimal exception occurs. If no overflow occurs, the XAD instruction resets CBIT to 0 to indicate success.

If a decimal exception occurs and bit 11 of the keys contains a 0, the instruction sets CBIT to 1. If bit 11 contains a 1, the instruction sets CBIT to 1 and causes a decimal exception fault. See Chapter 10 of the System Architecture Reference Guide for more information.

The registers used are GR0, GR1, GR3, GR4, GR6, FAR0, FAR1, FLR0, and FLR1. At the end of the instruction, the contents of these registers are indeterminate. The value of LINK is also indeterminate. The condition codes reflect the state of F2 after the decimal operation. (See Appendix A.)

► XBTD  
Binary to Decimal Conversion  
0 0 0 0 0 0 1 0 0 1 1 0 0 1 0 1

Converts a binary number to a decimal number. FAR0 contains the decimal field address. GR2 contains the control word. This instruction uses fields A, E, and H of the control word. H specifies the length of the binary number and its location, as follows:

<u>H</u>	<u>Length</u>	<u>Location</u>
0	16 bits	GR3 register, high side
1	32 bits	GR3 register
2	64 bits	DAC1 register

Converts the specified binary integer to a decimal integer and stores the result in the location specified by FAR0. Leaves the values of LINK indeterminate. Overflow results in a decimal exception. If no overflow occurs, resets CBIT to 0. The values of the condition codes are indeterminate.

The registers used are GR0, GR1, GR3, GR4, GR6, FAR0, and FLR0. At the instruction's end, the contents of the registers are indeterminate.

When the source register contains all zeros, the destination register will contain all zeros.

If a decimal exception occurs and bit 11 of the keys contains a 0, the instruction sets CBIT to 1. If bit 11 contains a 1, the instruction sets CBIT to 1 and causes a decimal exception fault. See Chapter 10 of the System Architecture Reference Guide for more information.

#### Note

The XBTB instruction does not use or modify FAR1, FLR1, or FAC1.

► XCM  
Decimal Compare  
0 0 0 0 0 0 1 0 0 1 0 0 0 0 1 0

Compares two decimal numbers and sets the condition codes depending on the result of the compare. Uses the G field of the control field to adjust the two numbers before the compare, as follows:

<u>G Field</u>	<u>Decision</u>
>0	Low-order digits of F1 only affect the initial borrow from the low-order digit of F2.
<0	Assume F1 is zero-extended with low zeros.

FAR0 contains the address of field 1 (F1). FAR1 contains the address of field 2 (F2). GR2 contains the control word. This instruction uses fields A, B, C, E, F, G, and H of the control word.

The registers used are GR0, GR1, GR3, GR4, GR6, FLR0, and FLR1. At the end of this instruction, the contents of these registers are indeterminate. When there is no decimal exception, CBIT is reset to 0. (This instruction cannot cause a decimal exception.) Leaves the value of LINK indeterminate. The condition codes reflect the result of the comparison, as follows.

<u>CC</u>	<u>Test Result</u>
GT	F2 > F1
EQ	F2 = F1
LT	F2 < F1

► **XDTB**  
 Decimal to Binary Conversion  
 0 0 0 0 0 0 1 0 0 1 1 0 0 1 1 0

Converts a decimal string to a binary string. FAR0 contains the address of the decimal string. GR2 contains the control word.

This instruction uses the A, E, and H fields. Field H specifies the length of the binary string and its location, as shown below.

<u>H</u>	<u>Length</u>	<u>Destination Register</u>
00	16 bits	GR2H
01	32 bits	GR2
10	64 bits	GR2/GR3

Converts the decimal string to a binary string of the specified type and stores it in the specified register. A conversion error causes a decimal exception. If no decimal exception occurs, the instruction sets CBIT to 0. The values of LINK and the condition codes are indeterminate.

The registers used are GR0, GR1, GR3, GR4, GR6, FAR0, and FLR0. At the end of the instruction, the contents of these registers are indeterminate.

If a decimal exception occurs and bit 11 of the keys contains a 0, the instruction sets CBIT to 1. If bit 11 contains a 1, the instruction sets CBIT to 1 and causes a decimal exception fault. See Chapter 10 of the System Architecture Reference Guide for more information.

#### Note

This instruction does not use or modify FAR1, FLR1, or FAC1.

► **XDV**  
 Decimal Divide  
 0 0 0 0 0 0 1 0 0 1 0 0 0 1 1 1

Divides a decimal number, D2, by another, D1, and stores the quotient and remainder in the location of D2.

FAR0 contains the address of D1. FAR1 contains the address of D2. L contains the control word. This instruction uses fields A, B, C, E, F, and H.

Both decimal numbers must be in trailing sign embedded format. In addition, D2 must contain a number of leading zeros equal to the length of D1.

The XDV instruction divides the two numbers. After the divide, the location of D2 contains the quotient of length (D2 length - D1 length) followed by the remainder of length (D1 length). Since D2 had leading zeros, no overflow can occur.

If the T bit contains a 1, the results will be forced positive. For more information about decimal arithmetic, refer to Chapter 6 of the System Architecture Reference Guide.

The registers used are GR0, GR1, GR3, GR4, GR6, FAR0, FAR1, FLR0, and FLR1. At the end of the instructions, the contents of these registers are indeterminate.

At the end of the instruction, the condition codes, LINK, FAR0, and FAR1 contain undefined results. If no overflow occurs, CBIT is reset to 0.

If D1 is 0, overflow occurs and causes a decimal exception. Decimal exceptions also occur if D1 or D2 has the incorrect data type or if the length of D2 is less than that of D1. If no decimal exception occurs, the instruction resets CBIT to 0.

If a decimal exception occurs and bit 11 of the keys contains a 0, the instruction sets CBIT to 1. If bit 11 contains a 1, the instruction sets CBIT to 1 and causes a decimal exception fault. See Chapter 10 of the System Architecture Reference Guide for more information.

► XED  
 Numeric Edit  
 0 0 0 0 0 0 1 0 0 1 0 0 1 0 1 0

Edits the contents of a string under control of a subprogram. The registers used are GR2, XB, FAR0, FAR1, and FLR0. At the end of the instruction, the contents of these registers and the CBIT, LINK, and condition codes are indeterminate.

FAR0 contains the address of the source string. The source string must be leading separate sign type and must have at least the same number of decimal digits and the decimal point alignment as called for in the edit subprogram.

FAR1 contains the address of the destination string. Bits 1 to 8 of GR2 contain the floating character; bits 9 to 16, the status register. Bits 17 to 24 of GR2 contain the number of remaining bytes to be processed (used if a fault or interrupt occurs). Bits 25 to 32 of GR2 contain the suppression character whose initial value is determined by bit 12 of the keys ('240 if bit 12 contains 0; '40 if bit 12 contains 1). XB contains the address of the edit subprogram.

The instruction uses an edit subprogram to alter a source string and store the edit result in a destination location(s). To set up, perform a decimal move to correct the type, alignment, and length of the number to be edited. Next, use a ICEQ instruction to set up the initial contents of the register.

Each 16-bit halfword in the edit subprogram has the format shown in Figure 3-4, where:

L is 1 if this 16-bit halfword is the last halfword  
in the subprogram,  
0 if it is not the last halfword;  
E is a suboperator;  
M is a suboperator modifier.

1	2	3	4	8	9	16
L	00		E		M	

Edit Subprogram Halfword Format  
Figure 3-4

The XED instruction uses several variables internally to control the edit subprogram. These are shown in Table 3-14. There are 17 edit suboperators, shown in Table 3-15.

Table 3-14  
XED Internal Variables

Var	Definition
SC	Zero suppression character; contained in B. Initial value is the space character ('240 or '40 if bit 12 of the keys contains 0 or 1, respectively).
FC	Floating edit character; contained in GR2. Initial value is not defined.
SIGN	Sign of the source field. The first character fetch sets up the value of this variable.
SIG	End zero suppression flag.

Table 3-15  
XED Suboperators

Subop	Mnem	Name and Description
00	ZS	Zero Suppress. Fetches M digits from the source field consecutively, each time checking SIG. If SIG is 1, copies the digit into the destination string. If SIG is 0 and the digit is not 0, inserts the floating character (if defined) and copies the digit into the destination field. If SIG is 0, the digit is not 0, and the floating character is not defined, sets the SIG flag and copies the digit into the destination. If SIG and the digit are both 0, substitutes SC for the 0 digit in the destination field.
01	IL	Insert Literal. Copies M into the destination string. Increments XB and FAR1 by 1.
02	SS	Set Suppress Character. Sets SC to M and increments XB by 1.
03	ICS	Insert Character. If SIG is 1, copies M into the destination string. If SIG is 0, copies SC into the destination string. Increments XB and FAR1 by 1.
04	ID	Insert Digits. If SIG is 0, and FC is defined, copies FC and M digits into the destination field then sets SIG to 1. Increments XB by 1, FAR0 by M, and FAR1 by M+1. If SIG is 0 and FC is not defined, sets SIG to 1 and copies M digits from the source to the destination. Increments XB by 1 and both FAR0 and FAR1 by M. If SIG is 1, copies M digits from the source to the destination and increments XB by 1 and both FAR0 and FAR1 by M.

Table 3-15 (continued)  
XED Suboperators

Subop	Mnem	Name and Description
05	ICM	Insert Character if Minus. If SIGN = 0, copies M into the destination string. If SIGN = 1, copies SC into the destination string. Increments both SB and FAR1 by 1.
06	ICP	Insert Character if Plus. If SIGN = 0, copies M into the destination string. If SIGN = 1, copies SC into the destination string. Increments both SB and FAR1 by 1.
07	SFC	Set Floating Character. Sets FC to M and increments XB by 1.
10	SFP	Set Floating if Plus. If SIGN = 0, sets FC to M. If SIGN = 1, FC to SC. Increments XB by 1.
11	SFM	Set Floating if Minus. If SIGN = 1, sets FC to M. If SIGN = 0, sets FC to SC. Increments XB by 1.
12	SFS	Set Floating to SIGN. If SIGN = 0, sets FC to '253. If SIGN = 1, sets FC to '255. Increments XB by 1.
13	JZ	Jump if Zero. If the condition flag in A = 0, increments XB by 1. If the condition flag in A = 1, adds M to XB and then increments XB by 1.
14	FS	Fill with Suppression Characters. Copies SC M times into the destination string. Increments XB by 1 and FAR1 by M.
15	SF	Set Significance. If SIG = 0 and FC <> 0, inserts FC into the destination string, sets SIG to 1, and increments XB and FAR1 by 1. If SIG = 0 and FC = 0, sets SIG to 1 and increments XB and FAR1 by 1. If SIG = 1, increments XB by 1.
16	IS	Insert Sign. If SIGN = 0, copies '253 into the destination string. If SIGN = 1, copies '255 into the destination string. Increments XB by 1.
17	SD	Suppress Digits. Fetches M digits from the source string and checks if they are '260. If the source digit = '260, inserts SC into the destination string. If the source digit <> '260, copies the source digit into the destination string. Increments XB by 1 and both FAR0 and FAR1 by M.
20	EBS	Embed Sign. Fetches M digits from the source string. If SIGN = 0, copies each digit into the destination string. If SIGN = 1, embeds a minus sign into each digit before copying it into the destination string. Table 6-15 shows the characters used to represent the sign/digit combinations. A } symbol represents negative 0.

► XH r,address  
 Exclusive OR Halfword  
 1 0 1 0 1 1 DR\3 TM\2 SR\3 BR\2  
 [ DISPLACEMENT\16 ]

Calculates an effective address, EA. Performs an exclusive OR of the contents of the specified r with the 16-bit value contained in the location specified by EA. Stores the result in r. Leaves the values of CBIT, LINK, and the condition codes unchanged.

#### Note

This instruction also has a register-to-register and an immediate form. See Appendix B for more information.

► XMP  
 Decimal Multiply  
 0 0 0 0 0 0 1 0 0 1 0 0 0 1 0 0

Multiplies one decimal number, M, by another, D1, and stores the result in D2's location in memory. M is right justified in field D2 at the start of the operation.

FAR0 contains the address of D1. FAR1 contains the address of D2. GR2 contains the control word; this instruction uses fields A, B, C, E, F, G, H, and T. Field G, the scale differential, must contain the number of decimal digits in M.

The number of decimal digits in D2 is greater than or equal to the number of decimal digits in D1 plus the number of decimal digits in M (specified by G). Normally, the digits to the left (more significant side) of M are zeros. If this is not the case, then a partial product field is added in.

The instruction multiplies M by D1 and stores the result in the location specified by FAR1. The result of the multiply is:

$D1 \times M + \text{partial product field}$

The partial product field is equal to:

$\text{length}(D2) - M.$

The partial product field is left justified in D2's location. The maximum partial product added in per traverse of the multiplicand is:

$\text{source digits} + \text{multiplier digits processed}$

There is also an implied weighting of the partial product field. The weighting is:

10 \*\* multiplier digits

If the T bit contains a 1, the results are forced positive.

The registers used are GR0, GR1, GR3, GR4, GR6, FAR0, FAR1, and XB. At the end of this instruction, the contents of these registers are indeterminate. At the end of the XMP instruction, the condition codes reflect the state of the result. (See Appendix A.) Overflow causes a decimal exception. If no overflow occurs, XMP resets CBIT to 0. LINK contains undefined results.

A decimal exception occurs if there are more potential or actual product digits than there is space in D2. If a decimal exception occurs and bit 11 of the keys contains a 0, the instruction sets CBIT to 1. If bit 11 contains a 1, the instruction sets CBIT to 1 and causes a decimal exception fault. See Chapter 10 of the System Architecture Reference Guide for more information.

► XMV  
Decimal Move  
0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 1

Moves a string of characters from one location to another.

FAR0 contains the address of the source string. FAR1 contains the address of the destination string. GR2 contains the control word. This instruction uses fields A, B, D, E, F, G, H, and T.

The instruction moves the contents of the source field into the destination field from right to left. If the B field in the control word is 1, the instruction changes the sign of the source field during the move. If the D field in the control word is 1 and the scale differential is greater than 0, the instruction rounds the source field during the move. If the scale differential (from the H field) is less than 0, the instruction pads the source field with SD trailing zeros before transferring.

If the T bit is set to 1, the result will be forced positive.

An overflow causes a decimal exception. If no decimal exception occurs, the instruction resets CBIT to 0. At the end of the instruction, LINK, FAR0, and FAR1 contain undefined results. The values of the condition codes reflect the state of the destination field after the move. (See Appendix A.)

If a decimal exception occurs and bit 11 of the keys contains a 0, the XMV instruction sets CBIT to 1. If bit 11 contains a 1, the instruction sets CBIT to 1 and causes a decimal exception fault. See Chapter 10 of the System Architecture Reference Guide for more information about decimal exceptions.

### Note

The source and destination strings may not overlap in memory.

► ZCM  
 Compare Character Field  
 0 0 0 0 0 0 1 0 0 1 0 0 1 1 1 1

Compares two fields and sets the condition codes depending on the result of the compare. Uses registers GR3, GR4, FAR0, FAR1, FLR0, and FLR1. At the end of this instruction, the contents of these registers are indeterminate.

FAR0 contains the address of field 1 (F1). FLR0 contains an integer specifying the length of F1. FAR1 contains the address of field 2 (F2). FLR1 contains an integer specifying the length of F2.

The instruction compares the contents of F1 and F2 on a byte by byte basis. If the fields are not of equal length, the instruction automatically extends the shorter string with space characters. Sets the condition codes as a result of the comparison, as follows:

<u>Result of Compare</u>	<u>Set Condition Codes</u>
F1 > F2	GT
F1 = F2	EQ
F1 < F2	LT

When the instruction completes execution, the values of CBIT and LINK are indeterminate.

#### Note

This instruction uses GR3, GR4, the FARs, and the FLRs during its operation. Since ZCM does not save the contents of these registers before using them, any data contained in them is overwritten when this instruction executes, unless you save it ahead of time.

► ZED  
 Character Field Edit  
 0 0 0 0 0 0 1 0 0 1 0 0 1 0 0 1

Controls an edit subprogram.

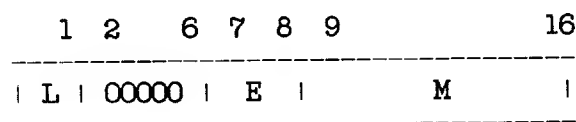
Uses the registers GR3, GR4, FAR0, FAR1, and FLR0. At the end of this instruction the contents of these registers are indeterminate. Leaves the values of CBIT, LINK, and the condition codes indeterminate.

FAR0 contains the address of the source string. FLR0 specifies the length of the source string. FAR1 contains the address of the

destination string. XB contains the address of the edit subprogram.

The ZED instruction uses the edit subprogram to alter the source string, then loads the edited result into the destination string. The subprogram, addressed by the contents of XB, contains a list of commands, each with the format shown in Figure 3-5, where:

L is 1 if this command is the last command in the subprogram,  
 0 if it is not;  
 E is the edit opcode;  
 M is the edit modifier.



ZED Subprogram Word Format  
Figure 3-5

Bits 2 to 6 must be 0.

M, the operator modifier, specifies information E uses when editing the source string. (See Table 3-16.)

E, the edit suboperator, specifies the operation to be performed on the source string. Table 3-16 shows the available values for E.

Table 3-16  
ZED Suboperators

Subop	Value	Action
CPC	00	Copies characters from the source string into the destination string. If the length of the source string is greater than the contents of the M field, then CPC moves a total of M source characters into the destination string, increments FAR0 and FAR1 by M, increments XB by 1, and decrements FLRO by M. If the length of the source string is less than the contents of the M field, then CPC moves the rest of the source string into the destination string, and then pads the remaining space to be filled with spaces. (See note.) Increments FAR0 by FLRO and FAR1 by M, increments XB by 1, and decrements FLRO by FLRO (so FLRO = 0).
INL	01	Inserts M into the destination string and increments both XB and FAR1 by 1.
SKC	10	Skips characters in the source string. If the remaining length of the source string is greater than or equal to the contents of the M field, then SKC skips over the next M characters of the source field by incrementing FAR0 by M and decrementing FLRO by M. If the remaining length of the source string is less than the contents of the M field, SKC skips over FLRO characters in the source string by incrementing FAR0 by FLRO and decrementing FLRO by FLRO (FLRO = 0). In either case, SKC increments XB by 1.
BLK	11	Inserts M spaces (see note) into the destination string, increments FAR1 by M, and increments XB by 1.

Note

A space is '240 or '40, depending on whether bit 12 of the keys is 0 or 1. This instruction uses GR3, GR4, the FARs, and the FLRs during its operation. Since ZED does not save the contents of these registers before using them, any data contained in them is overwritten when this instruction executes, unless you save it ahead of time.

► ZFIL  
 Fill Field with Character  
 0 0 0 0 0 0 1 0 0 1 0 0 1 1 1 0

Stores a character into a series of destination bytes. Uses registers GR3, GR4, FAR0, FAR1, FLR0, and FLR1. At the end of this instruction, the contents of these registers are indeterminate.

Bits 9 to 16 of GR2 contain the character to be stored. FAR1 contains the starting address of the destination field (byte aligned). FLR1 contains an integer specifying the length of the destination field (in bytes).

The instruction stores the character specified in GR2 in each byte of the destination field. If FLR1 contains 0, no operation takes place. Leaves the values of CBIT, LINK, and the condition codes indeterminate.

#### Note

This instruction uses GR3, GR4, the FARs, and the FLRs during its operation. Since ZFIL does not save the contents of these registers before using them, any data contained in them is overwritten when this instruction executes, unless you save it ahead of time.

► ZM address  
 Zero Memory Fullword  
 1 0 0 1 1 0 0 1 1 TM\2 SR\3 BR\2  
 DISPLACEMENT\16

Calculates an effective address, EA. Loads 0 into the 32-bit location specified by EA. Leaves the values of CBIT, LINK, and the condition codes unchanged.

► ZMH address  
 Zero Memory Halfword  
 1 0 1 1 1 0 0 1 1 TM\2 SR\3 BR\2  
 DISPLACEMENT\16

Calculates an effective address, EA. Loads 0 into the 16-bit location specified by EA. Leaves the values of CBIT, LINK, and the condition codes unchanged.

► ZMV  
 Move Character Field  
 0 0 0 0 0 0 1 0 0 1 0 0 1 1 0 0

Moves a character field from one location to another. Uses registers GR3, GR4, FAR0, FAR1, FLRO, and FLR1. At the end of this instruction, the contents of these registers are indeterminate.

FAR0 contains the address of the source string (byte aligned). FLRO specifies the length in bytes, N, of the source string. FAR1 contains the address of the destination string (byte aligned). FLR1 specifies the length in bytes, M, of the destination string.

Compares N and M. If N is less than M, the instruction moves the contents of the source string into the destination string followed by M-N space characters. A space character is '240 or '40 when bit 12 of the keys is 0 or 1, respectively. If the destination string is shorter, the instruction moves the first M characters of the source string into the destination string.

When the instruction completes, the values of FAR0, FAR1, FLRO, FLR1, CBIT, LINK, and the condition codes are indeterminate.

#### Note

This instruction uses GR3, GR4, the FARs, and the FLRs during its operation. Since ZMV does not save the contents of these registers before using them, any data contained in them is overwritten when this instruction executes, unless you save it ahead of time.

This instruction does not work with overlapping strings. See Chapter 6 of the System Architecture Reference Guide for more information.

► ZMVD  
 Move Characters Between Equal Length Strings  
 0 0 0 0 0 0 1 0 0 1 0 0 1 1 0 1

Moves characters from one string to another of equal length. Uses registers GR3, GR4, FAR0, FAR1, FLRO, and FLR1. At the end of this instruction, the contents of these registers are indeterminate.

FAR0 contains the address of the source string. FAR1 contains the address of the destination string. FLR1 contains the number of characters to move, N.

The instruction moves N characters from the source string to the destination string. Characters are moved from lower addresses to higher addresses.

When the ZMVD instruction completes, the values of FAR0, FAR1, FLR0, FLR1, CBIT, LINK, and the condition codes are indeterminate.

#### Note

The ZMVD instruction uses GR3, GR4, the FARs, and the FLRs during its operation. Since ZMVD does not save the contents of these registers before using them, any data contained in them is overwritten when this instruction executes, unless you save it ahead of time.

This instruction does not work with overlapping strings. See Chapter 6 of the System Architecture Reference Guide for more information.

► ZTRN  
Character String Translate  
0 0 0 0 0 0 1 0 0 1 0 0 1 0 0 0

Translates a string of characters and stores the translations in the specified destination. Uses registers GR3, GR4, FAR0, FAR1, FLR0, and FLR1. At the end of this instruction, the contents of these registers are indeterminate.

FAR0 contains the address of the source string (byte aligned). FAR1 contains the address of the destination string (byte aligned). FLR1 specifies the length of the source and destination strings. XB contains the address of a translation table. Each byte in the 256-byte table contains an alphabetic character.

The instruction uses the address in FAR0 to reference a character. It interprets this character as an integer, adding it to the contents of XB to form an address into the translation table. The instruction takes the referenced character in the translation table and writes it into the location specified by FAR1. After storing the character, the instruction increments the contents of FAR0 and FAR1 by 1, decrements the contents of FLR1 by 1, and repeats the operation until FLR1 contains 0.

At the end of the instruction, FAR0 and FAR1 point to the location that follows the last byte of the source and destination strings, respectively. FLR1 contains 0. Leaves the values of XB, CBIT, LINK, and the condition codes unchanged.

Note

This instruction uses GR3, GR4, the FARs, and the FLRs during its operation. Since ZTRN does not save the contents of these registers before using them, any data contained in them is overwritten when this instruction executes, unless you save it ahead of time.

# APPENDICES

# A

## Condition Code Information

Bits 9-10 of the keys contain the condition codes. Many arithmetic, branch, skip, jump, and other instructions set these bits to indicate the result of a test (result is less than 0, for example), to indicate whether a value is positive or negative, and so on. Other instructions use the condition code values as Boolean values. The instruction entries in Chapters 2 and 3 of this manual also describe how an instruction affects the state of these bits.

The LT condition code (bit 9 of the keys) contains the extended sign for arithmetic and comparison operations. The extended sign is the sign of the result as if the operation had been done on a machine of infinite precision; thus, LT shows the correct sign of the result despite any overflow. For logic operations, LT reflects the sign of the result.

The EQ condition code (bit 10 of the keys) shows whether or not a 16- or 32-bit result is equal to 0.

Table A-1 shows condition code interpretation for comparison, arithmetic, and logic operations.

Table A-1  
Interpretation of Condition Codes

LT, EQ Value	Comparison	Arithmetic	Logic
00	Register > 0 Register > EA Reg 1 > Reg 2	Signed result > 0 Unsigned result <> 0	Result <> 0, High-order bit = 0
01	Register = 0 Register = EA Reg 1 = Reg 2	Result = 0	Result = 0, High-order bit = 0
10	Register < 0 Register < EA Reg 1 < Reg 2	Result < 0	Result <> 0, High-order bit = 1
11	Not working	Possible if largest negative number is added to itself. (CBIT is set to 1 as well, to indicate overflow.)	Not working

# B

## Addressing Information

As noted in Chapter 1, the 50 Series processors support several kinds of addressing: direct addressing, indexed addressing, indirect addressing, indirect indexed addressing, and general register relative addressing. In addition, these processors also have several modes of addressing, each of which forms addresses differently.

### ADDRESSING MODES AND FORMATS

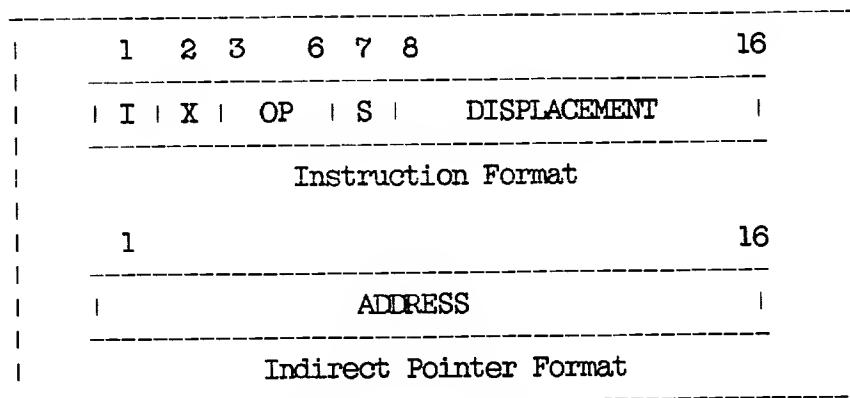
The addressing modes are listed below. Their formats and address formation are supplied in this Appendix.

- 64V Mode, Short Form
- 64V Mode, Long Form and Indirect Form
- 32I Mode
- 32R Mode
- 64R Mode
- 16S Mode
- 32S Mode

Address trap information is also provided at the end of this Appendix.

## 64V Mode Short Form

Figure B-1 and Table B-1 display and explain 64V mode short form instructions.



64V Mode Formats, Short Form  
Figure B-1

Table B-1  
64V Mode Short Form Summary

I	X	S	Disp	Inst Type	Example	Form of EA
0	0	0	0-'7@	Direct	LDA ADR	REG
			'10-'377	Direct		SB+D
			'400-'777	Direct@@		LB+D
0	1	0	0-'7@	Indexed	LDA ADR,X	REG, if D+X<'7;@
			'10-'377	Indexed		SB+D+X, if D+X>'7@
			'400-'777	Indexed@@		SB+D+X
						LB+D+X
1	0	0	0-'7@	Indirect	LDA ADR,*	I(REG)
			'10-'777	Indirect		I(PB+D)
1	1	0	0-'7	Indirect, preindexed	LDA ADR,X*	I(REG), if D+X<'7;@
						I(PB+D+X), if D+X>'7@
			'10-'77	Indirect, preindexed	LDA ADR,X*	I(PB+D+X)
			'100-'777	Indirect, postindexed	LDA ADR,*1	I(PB+D)+X
0	0	1	'-340-' +377	Direct	LDA ADR	P+D
0	1	1	'-340-' +377	Indexed	LDA ADR,1	P+D+X
1	0	1	'-340-' +377	Indirect	LDA ADR,*	I(P+D)
1	1	1	'-340-' +377	Indirect, preindexed	LDA ADR,1*	I(P+D+X)

Notes to Table B-1

- @ This table assumes segmented mode (modals bit 14 = 1). For nonsegmented mode, the displacement range is 0 to '37, rather than 0 to '7. This means that the range '10 to '377 changes to '40 to '377 in nonsegmented mode. The range '400 to '777 remains unchanged.
- @@ In these address forms, the displacement offsets the contents of LB by '400 (bit 8=1). To compensate for this, set the contents of LB to the current value of the link frame minus '400. For example, if the segment number in LB is '4002 and the offset number in the displacement is '177400, the offset of '400 gives the location of the link frame as segment number '4002, offset number 0.

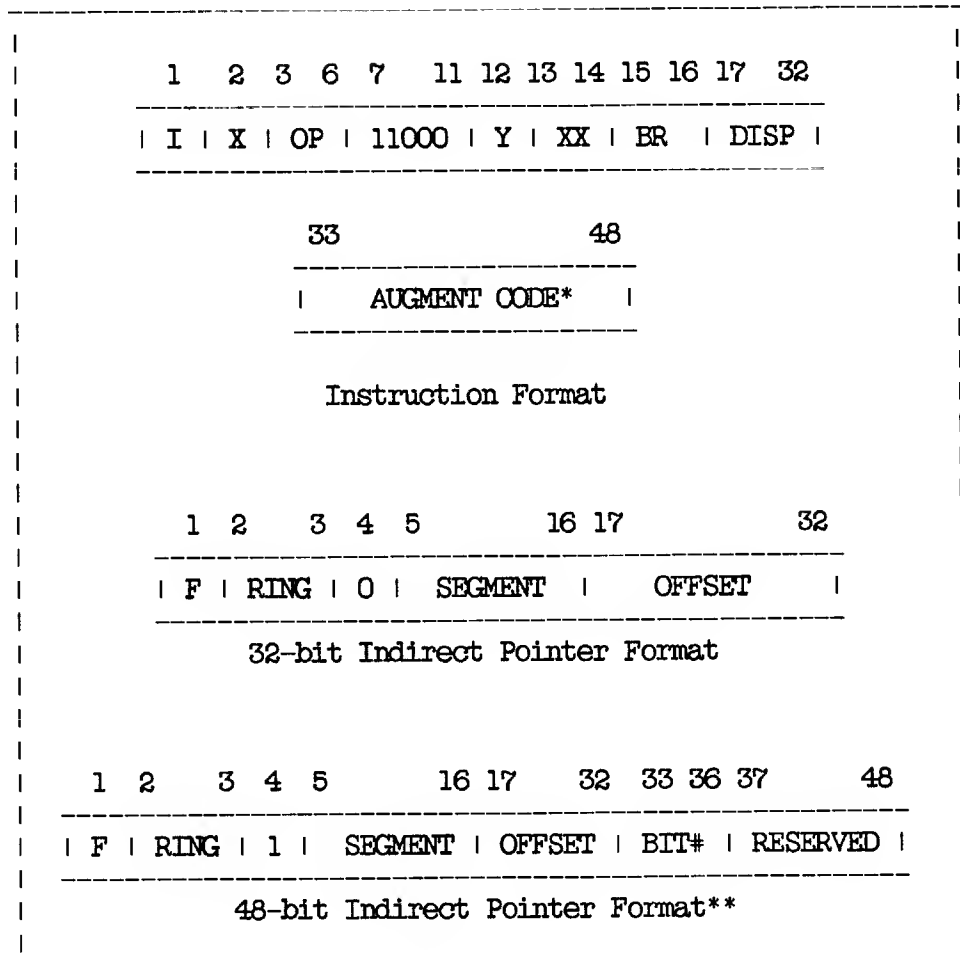
This mode allows one level of indexing, and one of indirection.

REG refers to a location in the register file. See Address Traps at the end of this chapter.

The instructions DFLX, FLX, JSX, LDX, LDY, QFLX, STX, and STY do not do indexing. The effective address is formed as if bit 2 = 0.

64V Mode, Long Form and Indirect Form

Figure B-2 and Table B-2 display and explain 64V mode long and indirect form instructions.



\* For quad operations only.

\*\* This indirect format is used only by a few instructions; most use the 32-bit form.

64V Mode Formats, Long Form and Indirect Form  
Figure B-2

Table B-2  
64V Mode Long Form, Indirect Summary

I	X	Y	BR	Instruction Type	Example	Form of EA
0	0	0	00	Direct	LDA ADR	PB/D
			01			SB+D
			10			LB+D
			11			XB+D
0	0	1	00	Indexed by Y	LDA ADR,Y	PB/D+Y
			01			SB+D+Y
			10			LB+D+Y
			11			XB+D+Y
0	1	0	00	Indexed by X	LDA ADR,X	PB/D+X
			01			SB+D+X
			10			LB+D+X
			11			XB+D+X
0	1	1	00	Indirect	LDA ADR,*	I(PB/D)
			01			I(SB+D)
			10			I(LB+D)
			11			I(XB+D)
1	0	0	00	Preindexed by Y	LDA ADR,Y*	I(PB/D+Y)
			01			I(SB+D+Y)
			10			I(LB+D+Y)
			11			I(XB+D+Y)
1	0	1	00	Postindexed by Y	LDA ADR,*Y	I(PB/D)+Y
			01			I(SB+D)+Y
			10			I(LB+D)+Y
			11			I(XB+D)+Y
1	1	0	00	Preindexed by X	LDA ADR,X*	I(PB/D+X)
			01			I(SB+D+X)
			10			I(LB+D+X)
			11			I(XB+D+X)
1	1	1	00	Postindexed by X	LDA ADR,*X	I(PB/D)+X
			01			I(SB+D)+X
			10			I(LB+D)+X
			11			I(XB+D)+X

#### Notes to Table B-2

The processor performs X and Y indexing and 32-bit word (inter-segment) indirection.

PB/D indicates that the displacement is relative to the origin of PB. PB specifies the segment number (the offset must be 0); the displacement specifies the offset.

All displacements are within the range 0 to '1777777.

The instructions DFLX, FLX, JSX, LDX, LDY, QFLX, STX, and STY do not do indexing. The effective address is formed as shown in Table B-3. Bit 2, the X bit, is used as part of the opcode in these instructions.

Table B-3  
Address Formation for Nonindexing Instructions

I	X	Y	Instruction Type
0	0	0	Direct
0	0	1	Direct
0	1	0	Direct
0	1	1	Direct
1	0	0	I(A)
1	0	1	I(A)
1	1	0	I(A)
1	1	1	I(A)

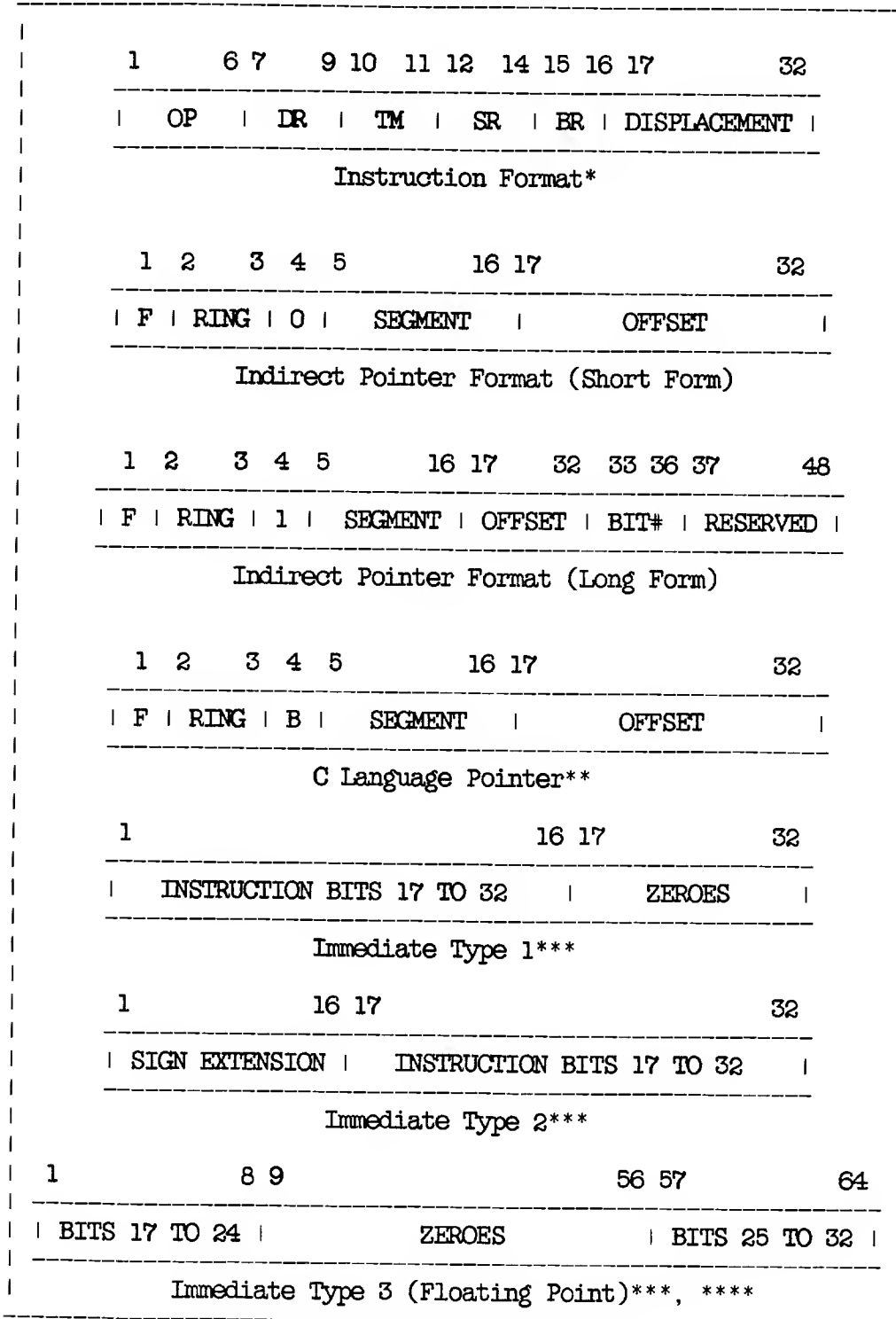
Notes to Table B-3

For the earlier processors listed in "About This Book", see Appendix B for information on their address formation for nonindexing instructions.

The symbol A in Table B-3 represents the value calculated from the base register (PB, SB, LB, or XB) and displacement in the instruction.

32I Mode

Figure B-3 and Table B-4 display and explain 32I mode instructions.



32I Mode Formats  
Figure B-3

Notes to Figure B-3

- \* TM is the tag modifier which, in combination with the SR and BR fields, specifies the instruction type.
- \*\* The C language pointer is not available for the earlier processors listed in "About This Book".
- \*\*\* The instruction specifies the immediate type to use. During instruction execution, the processor forms the immediate in the appropriate format and stores it internally for use in the operation as shown in Figure B-3.
- \*\*\*\* Bits 1 to 8 of Immediate Type 3 are formed from I mode instruction bits 17 to 24; bits 57 to 64 from I mode instruction bits 25 to 32.

Table B-4  
32I Mode Summary

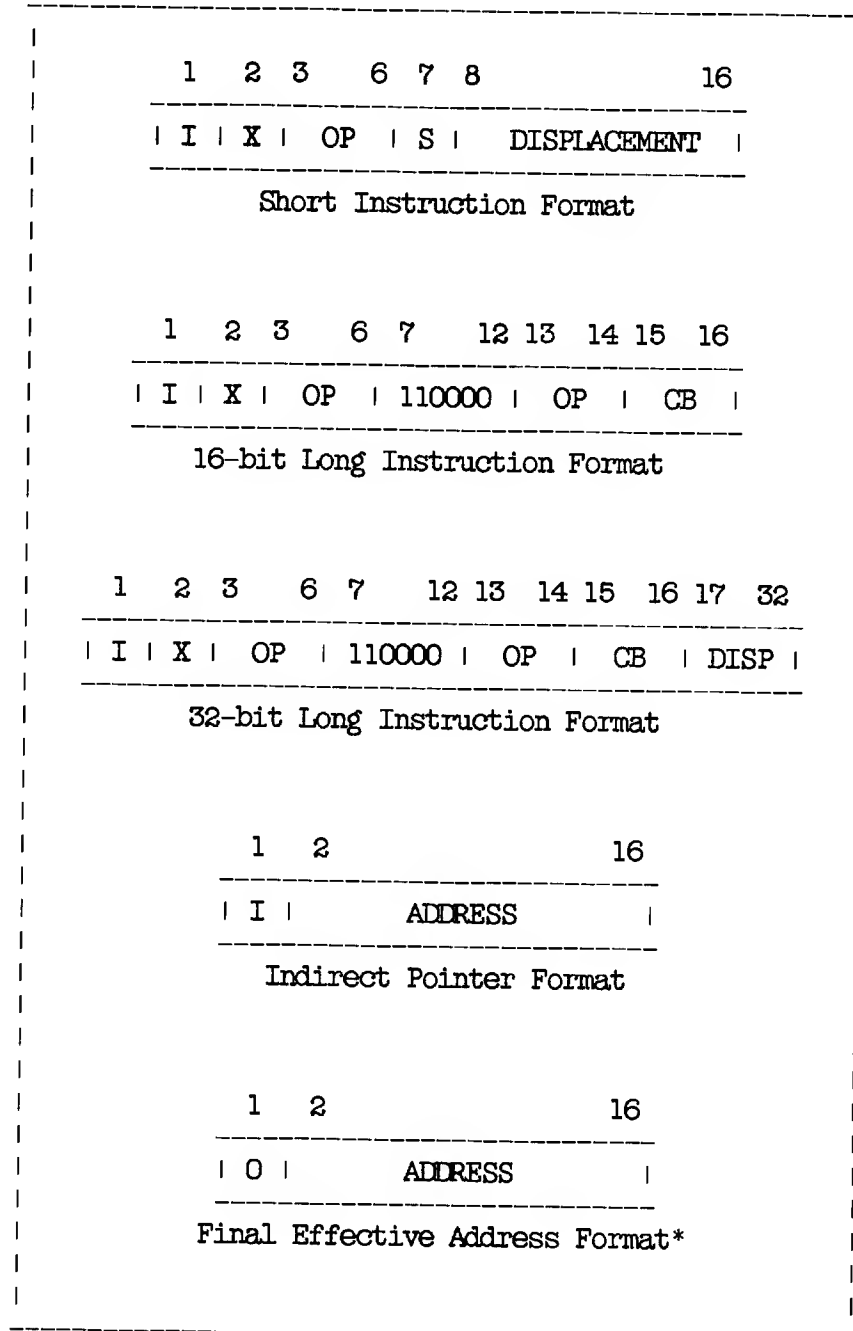
TM	SR	BR	Instruction Type	EA (Segment)	EA (Offset)
3	0	-	Indirect	I(5 to 16)	I(D+BR)
3	>0	-	Indirect postindexed	I(5 to 16)	(I(D+BR))+SRH
2	0	-	Indirect	I(5 to 16)	I(D+BR)
2	>0	-	Indirect preindexed	I(5 to 16)	I(D+BR+SRH)
1	0	-	Direct	BR(5 to 16)	D+BR
1	>0	-	Indexed	BR(5 to 16)	D+BR+SRH
0	0-7	0	Register-to-register	---	---
0	0	1	Immediate type 1	---	---
0	>0	1	Immediate type 2	---	---
0	0	2	Immediate type 3	---	---
0	1	2	Floating register source (FRO)	---	---
0	2	2	Undefined; generates UII (unimplemented instruction) fault	---	---
0	3	2	Floating register source (FR1)	---	---
0	4-7	2	Undefined; generates UII fault	---	---
0	0-7	3	General register relative (undefined for the earlier processors listed in "About This Book")	SR(5 to 16)	D+SRL

Note to Table B-4

Displacements are within the range 0 to '177777, inclusive.

32R Mode

Figure B-4 and Table B-5 display and explain 32R mode instructions.



32R Mode Formats  
Figure B-4

Note to Figure B-4

The final form of an effective address in 32R mode is only 15 bits wide. Special hardware exists to truncate the effective address to this length. The program counter, however, is a full 16 bits wide. Multilevel indirection is a feature of 32R mode.

Table B-5  
32R Mode Summary

I	X	S	CB	Displacement	Instruction Type	Form of EA
0	0	0	--	0 to '777	Direct	O/D
0	1	0	--	0 to '777	Indexed	O/D+X
1	0	0	--	0 to '777	Indirect	I(O/D)
1	1	0	--	0 to '77	Indirect, preindexed	I(O/D+X)
1	1	0	--	'100 to '777	Indirect, postindexed	I(O/D)+X
0	0	1	--	'-360 to '+377	Direct	P+D
0	1	1	--	'-360 to '+377	Indexed	P+D+X
1	0	1	--	'-360 to '+377	Indirect	I(P+D)
1	1	1	--	'-360 to '+377	Indirect postindexed	I(P+D)+X
0	0	1	2	---	@Postincrement	SP
0	1	1	2	---	@Postincrement, indirect, postindexed	I(SP)+X
1	0	1	2	---	@Postincrement, indirect	I(SP)
0	0	1	3	---	#Predecrement	SP-1
0	1	1	3	---	#Predecrement, indirect, postindexed	I(SP-1)+X
1	0	1	3	---	#Predecrement, indirect	I(SP-1)
0	0	1	0	0 to '1777777	*Direct, long reach	D
0	1	1	0	0 to '1777777	*Indexed, long reach	D+X
1	0	1	0	0 to '1777777	*Indirect, long reach	I(D)
1	1	1	0	0 to '1777777	*Indirect, preindexed, long reach	I(D+X)
1	1	1	2	0 to '1777777	*Indirect, postindexed, long reach	I(D)+X
0	0	1	1	0 to '1777777	*Direct, stack relative	D+SP
0	1	1	1	0 to '1777777	*Indexed, stack relative	D+SP+X
1	0	1	1	0 to '1777777	*Indirect, stack relative	I(D+SP)
1	1	1	1	0 to '1777777	*Indirect, preindexed stack relative	I(D+SP+X)
1	1	1	3	0 to '1777777	*Indirect, postindexed stack relative	I(D+SP)+X

Notes to Table B-5

\* These instruction types use the 32-bit long format shown in Figure B-4.

@ These instruction types use the 16-bit long format shown in Figure B-4. They also increment the contents of SP by 1 during EA formation.

# These instruction types use the 16-bit long format shown in Figure B-4. They also decrement the contents of SP by 1 during EA formation.

For all instruction types listed above, address traps can occur when any part of the EA formation results in an address in the range 0 to '7 (segmented mode) or 0 to '37 (unsegmented mode). See the end of this chapter for more information.

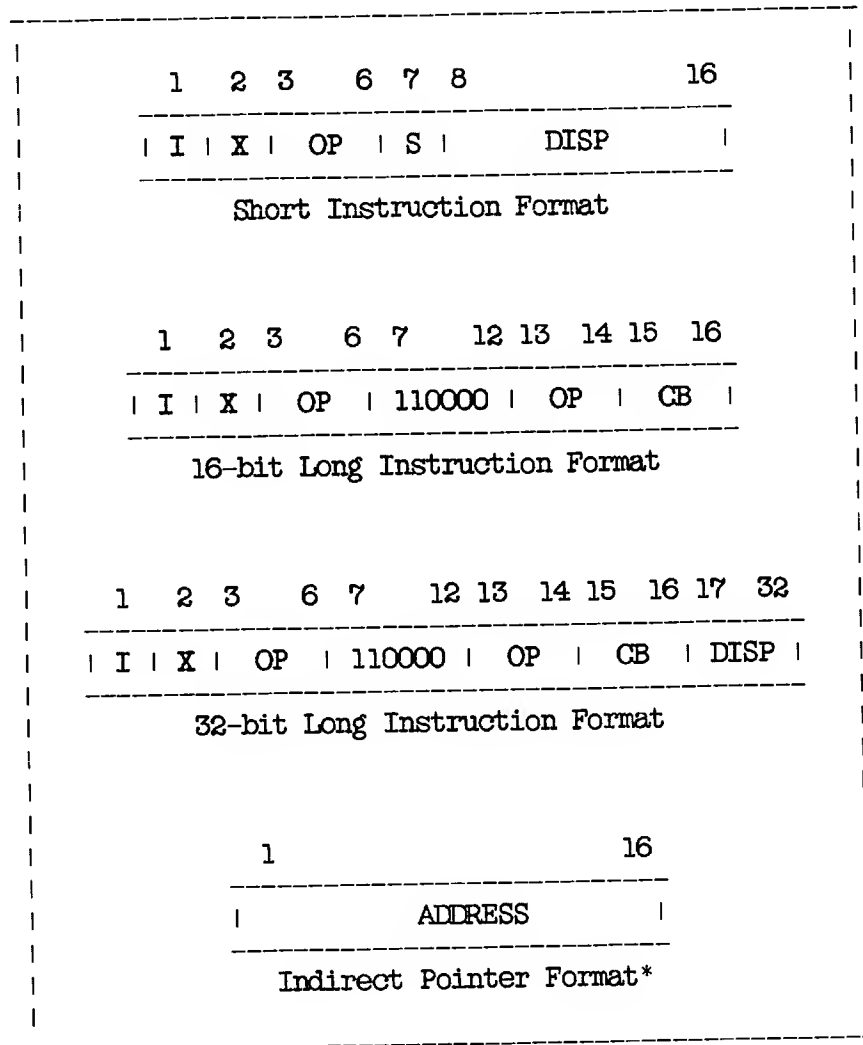
The processor performs one level of indexing and multiple levels of indirection.

O/D indicates that the displacement is within Sector 0.

The instructions DFLX, FLX, JSX, LDX, LDY, QFLX, STX, and STY do not do indexing. The processor treats the X bit as a 0 to determine what addressing mode to use. For example, if one of these instructions specifies I, X, S, and CB as 0113, the processor interprets it as 0013.

64R Mode

Figure B-5 and Table B-6 display and explain 64R mode instructions.



\*Only a single level of indirection is possible in 64R mode.

64R Mode Formats  
Figure B-5

Table B-6  
64R Mode Summary

I	X	S	CB	Displacement	Instruction Type	Form of EA
0	0	0	--	0 to '777	Direct	O/D
0	1	0	--	0 to '777	Indexed	O/D+X
1	0	0	--	0 to '777	Indirect	I(O/D)
1	1	0	--	0 to '77	Indirect, preindexed	I(O/D+X)
1	1	0	--	'100 to '777	Indirect, postindexed	I(O/D)+X
0	0	1	--	'-360 to '+377	Direct	P+D
0	1	1	--	'-360 to '+377	Indexed	P+D+X
1	0	1	--	'-360 to '+377	Indirect	I(P+D)
1	1	1	--	'-360 to '+377	Indirect postindexed	I(P+D)+X
0	0	1	2	---	@Postincrement	SP
0	1	1	2	---	@Postincrement, indirect, postindexed	I(SP)+X
1	0	1	2	---	@Postincrement, indirect	I(SP)
0	0	1	3	---	#Predecrement	SP-1
0	1	1	3	---	#Predecrement, indirect, postindexed	I(SP-1)+X
1	0	1	3	---	#Predecrement, indirect	I(SP-1)
0	0	1	0	0 to '177777	*Direct, long reach	D
0	1	1	0	0 to '177777	*Indexed, long reach	D+X
1	0	1	0	0 to '177777	*Indirect, long reach	I(D)
1	1	1	0	0 to '177777	*Indirect, preindexed, long reach	I(D+X)
1	1	1	2	0 to '177777	*Indirect, postindexed, long reach	I(D)+X
0	0	1	1	0 to '177777	*Direct, stack relative	D+SP
0	1	1	1	0 to '177777	*Indexed, stack relative	D+SP+X
1	0	1	1	0 to '177777	*Indirect, stack relative	I(D+SP)
1	1	1	1	0 to '177777	*Indirect, preindexed stack relative	I(D+SP+X)
1	1	1	3	0 to '177777	*Indirect, postindexed stack relative	I(D+SP)+X

Notes to Table B-6

For all the instruction types listed in Table B-6, address traps can occur when any part of the EA formation results in an address in the range 0 to '7 (segmented mode) or 0 to '37 (unsegmented mode). See the end of this chapter for more information.

\* These instruction types use the 32-bit long format shown in Figure B-5.

@ These instruction types use the 16-bit long format shown in Figure B-5. They also increment the contents of SP by 1 during EA formation.

# These instruction types use the 16-bit long format shown in Figure B-5. They also decrement the contents of SP by 1 during EA formation.

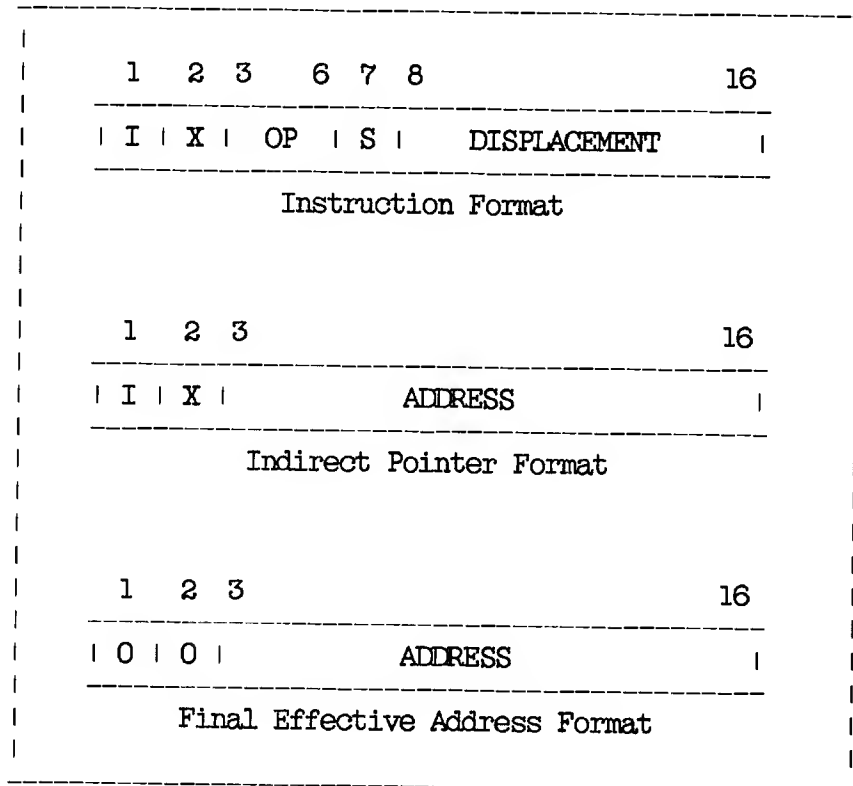
The processor performs one level of indexing and multiple levels of indirection.

O/D indicates that the displacement is within Sector 0.

The instructions DFLX, FLX, JSX, LDX, LDY, QFLX, STX, and STY do not do indexing. The processor treats the X bit as a 0 to determine what addressing mode to use. For example, if one of these instructions specifies I, X, S, and CB as 0113, the processor interprets it as 0013.

16S Mode

Figure B-6 and Table B-7 display and explain 16S mode instructions.



16S Mode Formats  
Figure B-6

Note to Figure B-6

The final form of effective addresses in S mode are only 14 bits wide. Special hardware exists to truncate the effective address to this length. The program counter, however, is a full 16 bits wide.

Table B-7  
16S Mode Summary

I	X	S	Disp	Instruction Type	Example	EA Form
0	0	0	0 to '777	Direct	LDA ADR	O/D
0	0	1	0 to '777	Direct	LDA ADR	C/D
0	1	0	0 to '777	Indexed	LDA ADR,1	O/D+X
0	1	1	0 to '777	Indexed	LDA ADR,1	C/D+X
1	0	0	0 to '777	Indirect	LDA ADR,*	I(O/D)
1	0	1	0 to '777	Indirect	LDA ADR,*	I(C/D)
1	1	0	0 to '777	Indirect preindexed	LDA ADR,1*	I(D+X)
1	1	1	0 to '777	Indirect preindexed	LDA ADR,1*	I(D+X)

Notes to Table B-7

The processor performs indexing before resolving each level of indirection.

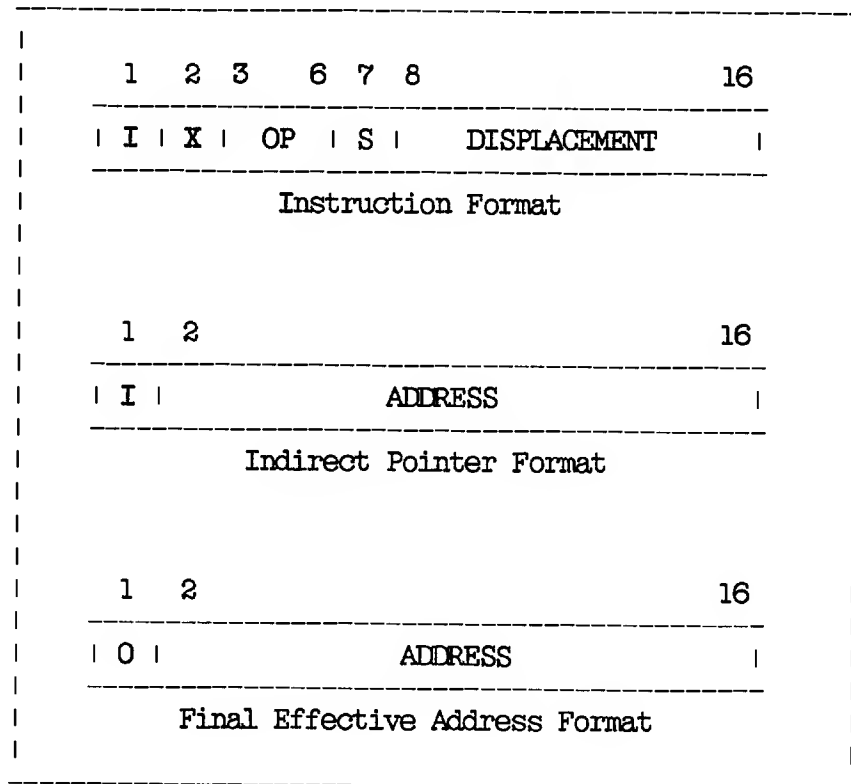
This mode allows multiple levels of both indexing and indirection.

The instructions, LDX and STX, cannot do indexing. The effective address is formed as if bit 2 = 0.

O/D indicates that the displacement is within Sector 0; C/D, within the current sector.

32S Mode

Figure B-7 and Table B-8 display and explain 32S mode instructions.



32S Mode Formats  
Figure B-7

Note to Figure B-7

The final form of effective addresses in S mode are only 15 bits wide. Special hardware exists to truncate the effective address to this length. The program counter, however, is a full 16 bits wide.

Table B-8  
32S Mode Summary

I	X	S	Disp	Instruction Type	Example	EA Form
0	0	0	0 to '777	Direct	LDA ADR	O/D
0	0	1	0 to '777	Direct	LDA ADR	C/D
0	1	0	0 to '777	Indexed	LDA ADR,1	O/D+X
0	1	1	0 to '777	Indexed	LDA ADR,1	C/D+X
1	0	0	0 to '777	Indirect	LDA ADR,*	I(O/D)
1	0	1	0 to '777	Indirect	LDA ADR,*	I(C/D)
1	1	0	0 to '77	Indirect preindexed	LDA ADR,1*	I(D+X)
1	1	0	'100 to '777	Indirect postindexed	LDA ADR,*1	I(D)+X
1	1	1	0 to '777	Indirect postindexed	LDA ADR,*1	I(D)+X

#### Notes to Table B-8

The processor performs indexing before resolving each level of indirection.

This mode allows one level of indexing, and multiple levels of indirection.

The instructions, LDX and STX, cannot do indexing. The effective address is formed as if bit 2 = 0.

#### ADDRESS TRAPS

Several of the summaries in the last section specify special cases of EA formation when the address is within a particular range. This range of addresses corresponds to registers within the current user register set in the register file. (See Chapter 9 of the System Architecture Reference Guide.) In segmented mode, this range is '0 to '7; in nonsegmented mode, '0 to '37. This range of addresses for segmented and nonsegmented modes is referred to as the ATR, or address trap range, throughout this section.

The registers within the user register set contain information, such as general, base, floating-point, and index registers, and system status and control information. Each time any part of the EA formation generates an address within the ATR, an address trap aborts any read or write to a memory location and instead references the specific register.

Table B-9 summarizes when address traps occur for all modes of addressing and instruction types.

Table B-9  
Address Trap Information

Mode	Inst Type	Action
16S 32S 32R 64R	Memory reference	Address trap occurs if the EA falls within the ATR (address trap range). The instruction format or length has no bearing.
	Generic	Address traps never occur.
	Generic AP	Address traps do not occur when the processor is fetching the address pointer.
64V	32-bit memory reference	Address traps never occur.
	Short format	See Table B-10.
	16-bit indirect	Address traps occur if the EA falls within the ATR.
	32-bit indirect	Address traps never occur.
32I	All types	Address traps never occur.

When bits 17 to 32 of the program counter contain a value within the ATR and the processor is reading an instruction, an address trap always occurs. The only exception to this is if the machine is operating in 32I mode.

When the processor executes short format instructions in 64V mode, address traps can occur during operand fetches or indirect fetches. Table B-10 lists the conditions that must be present for an address trap to occur.

Table B-10

Address Trap Action for Short Format  
Instructions, 64V Mode

I	X	S	Disp	Action
0	0	0	0 to '7	Takes address trap.
0	0	0	'10 to '37	Takes address trap only if segmentation is off.
0	0	0	'40 to '377	Cannot take address trap.
0	0	1	-'340 to +'377	Takes address trap if EA (P+D) is within the ATR.
0	1	0	0 to ATR	Takes address trap if D+X is within the ATR. If D+X is outside the ATR, the EA is SB (seg #)   D+X (for the 750, 850, and 2350 to 9955 II; or SB (seg #)   D+X+SB (offset #) (for all other machines).
			From ATR to '377	Cannot take address trap; EA is SB+D+X (for 750, 850, and 2350 to 9955 II).
				All other machines take address trap if D+X is within the ATR.
			'400 to '777	Cannot take address trap.
0	1	1	-'340 to +'377	Takes address trap if EA (P+D+X) is within the ATR.
1	0	0	0 to '777	Takes address trap if D is within the ATR.*
1	0	1	-'340 to +'377	Takes address trap if EA ( (P+D) ) is within the ATR.*
1	1	0	0 to '777	Takes address trap if D<'100 and D+X is within the ATR.*
1	1	1	-'340 to +'377	Takes address trap if EA (P+D) is within the ATR.*

Note to Table B-10

\* The indirect address also takes an address trap if EA is within the ATR.

If an instruction specifies a write operation that could potentially cause an address trap, the instruction loads the data to be written into a temporary register. If a trap occurs, the routine aborts the write to memory. It loads the specified register file location with the contents of the temporary register.

If the instruction specifies a read operation that causes an address trap, the trap routine aborts the memory read and fetches the contents of a register file location. The trap routine loads the cache from the register file data and allows the processor one cache access before invalidating the cache location.

Table B-11 shows the address trap locations and the registers to which they correspond. For more information on the register file, see Chapter 9 of the System Architecture Reference Guide.

Table B-11  
Address Trap/Register File Correspondence

AT	S and R Modes	V Mode
'0	X	X
'1	A	A, LH
'2	B	LL
'3	S	Y
'4	FAC bits 1 to 16	FAC bits 1 to 16
'5	FAC bits 17 to 32	FAC bits 17 to 32
'6	FAC exponent	FAC exponent
'7	PC, LSBs	PC, LSBs
'10*	DTAR3H	DTAR3H
'11*	FOODEH	FOODEH
'12*	FADDRL	FADDRL
'13*		
'14*		SEH
'15*		SEL
'16*		LEH
'17*		LEL
'20*	DMA cell '20H	DMA cell '20H
'21*	DMA cell '20L	DMA cell '20L
'22*	DMA cell '22H	DMA cell '22H
'23*	DMA cell '22L	DMA cell '22L
'24*	DMA cell '24H	DMA cell '24H
'25*	DMA cell '24L	DMA cell '24L
'26*	DMA cell '26H	DMA cell '26H
'27*	DMA cell '26L	DMA cell '26L
'30*	DMA cell '30H	DMA cell '30H
'31*	DMA cell '30L	DMA cell '30L
'32*	DMA cell '32H	DMA cell '32H
'33*	DMA cell '32L	DMA cell '32L
'34*	DMA cell '34H	DMA cell '34H
'35*	DMA cell '34L	DMA cell '34L
'36*	DMA cell '36H	DMA cell '36H
'37*	DMA cell '36L	DMA cell '36L

Note to Table B-11

- \* These correspond to user register file locations only in nonsegmented mode.

SUMMARY

The fields of a memory reference instruction specify information used to form an effective address. These fields specify which information is to be used in the formation, how the formation is to be done, and -- in conjunction with the rest of the program -- the addressing mode under which the address is to be formed. Depending on the segmentation mode and the EA formation, addresses can reference registers within the current user register file as well as memory locations.

# C

## Instruction Summary Charts

This appendix contains two instruction summary charts: one for S, R, and V modes; another for I mode. Each chart contains a list of instructions for the Prime 50 Series processors. (Appendix E lists those instructions that have been archived.) Each instruction is followed by its octal code, format, function, addressing mode, CBIT, LINK, and condition code information, and a one-line description of the instruction.

The columns in each chart are as follows:

R            Restrictions:

Blank	Regular instruction.
R	Instruction causes a restricted mode fault if executed in other than Ring 0.
P	Instruction may cause a fault depending on address.

Mnem        A mnemonic name recognized by the assembler PMA.

Opcode      Octal operation code portion of the instruction.

RI           Register (R) and Immediate (I) forms, if available.

## INSTRUCTION SETS GUIDE

Form      Format of instruction:

<u>Mnemonic</u>	<u>Definition</u>
AP	Address Pointer
BRAN	Branch
CHAR	Character
DECI	Decimal
GEN	Generic
GR	General Register -- non Memory Reference
IERN	I Mode Branch
MR	Memory Reference -- Non I Mode
MRFR	Memory Reference -- Floating Register
MRGR	Memory Reference -- General Register
MRNR	Memory Reference -- Non Register
PIO	Programmed I/O
RGEN	Register Generic
SHFT	Shift

Func      Function of instruction:

<u>Mnemonic</u>	<u>Definition</u>
ADMOD	Addressing Mode
BRAN	Branch
CHAR	Character
CLEAR	Clear Field
CPTR	C Language Pointer
DECI	Decimal Arithmetic
FIELD	Field Register
FLPT	Floating Point Arithmetic
GRR	General Register Relative
INT	Integer
INTGY	Integrity
IO	Input/Output
KEYS	Keys
LOGIC	Logical Operations
LTSTS	Logical Test and Set
MCTL	Machine Control
MOVE	Move
PCILJ	Program Control and Jump
PRCEX	Process Exchange
QUEUE	Queue Control
SHIFT	Register Shift
SKIP	Skip

M      Addressing modes of instructions:

<u>Mode</u>	<u>Name</u>
S	Sectored
R	Relative
V	Virtual (64V)
I	32I

C      How instruction affects the CBIT and LINK.

<u>Code</u>	<u>Definition</u>
-	CBIT and LINK are unchanged
1	CBIT = unchanged; LINK = carry
2	CBIT = overflow status; LINK = carry
3	CBIT = overflow status; LINK = indeterminate
4	CBIT = shift extension; LINK = shift extension
5	CBIT = result; LINK = indeterminate
6	CBIT and LINK are indeterminate
7	CBIT and LINK are loaded by the instruction
8	CBIT = result; LINK = unchanged
9	CBIT = unchanged; LINK = indeterminate
*	CBIT and LINK values vary among processors; see individual instruction description

CC      How instruction affects the condition codes.

<u>Code</u>	<u>Definition</u>
-	Condition codes are unchanged.
1	Condition codes are set to reflect the result of arithmetic operation or compare.
4	Condition codes are set to reflect result of branch, compare, or logicize operand state.
5	Condition codes are indeterminate.
6	Condition codes are loaded by instruction.
7	Condition codes show special results for this instruction.

Description    A brief description of the instruction.

Table C-1 contains a summary of S, R, and V mode instructions. Table C-2 is a summary of I mode instructions. Instructions that have been archived are not in either of these tables; see Appendix E for them.

Table C-1  
S, R, V Mode Instruction Summary

R	Mnem	Opcode	Form	Func	M	C	OC	Description
	A1A	141206	GEN	INT	SRV	2	1	Add One to A
	A2A	140304	GEN	INT	SRV	2	1	Add Two to A
	ABQ	141716	AP	QUEUE	V	-	7	Add Entry to Bottom of Queue
	ACA	141216	GEN	INT	SRV	2	1	Add CBIT to A
	ADD	06	MR	INT	SRV	2	1	Add
	ADL	06 03	MR	INT	V	2	1	Add Long
	ADLL	141000	GEN	INT	V	2	1	Add LINK to L
	ALFA 0	001301	GEN	FIELD	V	6	-	Add L to FAR 0
	ALFA 1	001311	GEN	FIELD	V	6	-	Add L to FAR 1
	ALL	0414XX	SHFT	SHIFT	SRV	4	-	A Left Logical
	ALR	0416XX	SHFT	SHIFT	SRV	4	-	A Left Rotate
	ALS	0415XX	SHFT	SHIFT	SRV	3	-	A Arithmetic Left Shift
	ANA	03	MR	LOGIC	SRV	-	-	AND to A
	ANL	03 03	MR	LOGIC	V	-	-	AND to A Long
	ARGT	000605	GEN	PCTLJ	V	6	5	Argument Transfer
	ARL	0404XX	SHFT	SHIFT	SRV	4	-	A Right Logical
	ARR	0406XX	SHFT	SHIFT	SRV	4	-	A Right Rotate
	ARS	0405XX	SHFT	SHIFT	SRV	4	-	A Arithmetic Right Shift
	ATQ	141717	AP	QUEUE	V	-	7	Add Entry to Top of Queue
	BCEQ	141602	BRAN	BRAN	V	-	-	Branch on Condition Code EQ
	BOGE	141605	BRAN	BRAN	V	-	-	Branch on Condition Code GE
	BOGT	141601	BRAN	BRAN	V	-	-	Branch on Condition Code GT
	BCLE	141600	BRAN	BRAN	V	-	-	Branch on Condition Code LE
	BCLT	141604	BRAN	BRAN	V	-	-	Branch on Condition Code LT
	BCNE	141603	BRAN	BRAN	V	-	-	Branch on Condition Code NE
	BCR	141705	BRAN	BRAN	V	-	-	Branch on CBIT Reset to 0
	BCS	141704	BRAN	BRAN	V	-	-	Branch on CBIT Set to 1
	BDX	140734	BRAN	BRAN	V	-	-	Branch on Decrement X
	BDY	140724	BRAN	BRAN	V	-	-	Branch on Decrement Y
	BEQ	140612	BRAN	BRAN	V	-	4	Branch on A Equal to 0
	BFEQ	141612	BRAN	BRAN	V	-	4	Branch on F Equal to 0
	BFGE	141615	BRAN	BRAN	V	-	4	Branch on F Greater Than or Equal to 0
	BFGT	141611	BRAN	BRAN	V	-	4	Branch on F Greater Than 0
	BFLE	141610	BRAN	BRAN	V	-	4	Branch on F Less Than or Equal to 0
	BFLT	141614	BRAN	BRAN	V	-	4	Branch on F Less Than 0
	BFNE	141613	BRAN	BRAN	V	-	4	Branch on F Not Equal to 0
	BGE	140615	BRAN	BRAN	V	-	4	Branch on A Greater Than or Equal to 0
	BGT	140611	BRAN	BRAN	V	-	4	Branch on A Greater Than 0
	BIX	141334	BRAN	BRAN	V	-	-	Branch on Incremented X
	BIY	141324	BRAN	BRAN	V	-	-	Branch on Incremented Y
	BLE	140610	BRAN	BRAN	V	-	4	Branch on A Less Than or Equal to 0
	BLEQ	140702	BRAN	BRAN	V	-	4	Branch on L Equal to 0

Table C-1 (continued)  
S, R, V Mode Instruction Summary

R	Mnem	Opcode	Form	Func	M	C	OC	Description
	BLGE	140615	BRAN	BRAN	V	-	4	Branch on L Greater Than or Equal to 0
	BLGT	140701	BRAN	BRAN	V	-	4	Branch on L Greater Than 0
	BLLE	140700	BRAN	BRAN	V	-	4	Branch on L Less Than or Equal to 0
	BLLT	140614	BRAN	BRAN	V	-	4	Branch on L Less Than 0
	BLNE	140703	BRAN	BRAN	V	-	4	Branch on L Not Equal to 0
	BLR	141707	BRAN	BRAN	V	-	-	Branch on LINK Reset to 0
	BLS	141706	BRAN	BRAN	V	-	-	Branch on LINK Set to 1
	BLT	140614	BRAN	BRAN	V	-	4	Branch on A Less Than 0
	BMEQ	141602	BRAN	BRAN	V	-	-	Branch on Magnitude Condition EQ
	BMGE	141706	BRAN	BRAN	V	-	-	Branch on Magnitude Condition GE
	BMGT	141710	BRAN	BRAN	V	-	-	Branch on Magnitude Condition GT
	BMLE	141711	BRAN	BRAN	V	-	-	Branch on Magnitude Condition LE
	BMLT	141707	BRAN	BRAN	V	-	-	Branch on Magnitude Condition LT
	BMNE	141603	BRAN	BRAN	V	-	-	Branch on Magnitude Condition NE
	BNE	140613	BRAN	BRAN	V	-	4	Branch on A Not Equal to 0
	CAL	141050	GEN	CLEAR	SRV	-	-	Clear A Left Byte
	CALF	000705	AP	PCTLJ	V	6	5	Call Fault Handler
	CAR	141044	GEN	CLEAR	SRV	-	-	Clear A Right Byte
	CAS	11	MR	SKIP	SRV	1	1	Compare A and Skip
	CAZ	140214	GEN	SKIP	SRV	1	1	Compare A with 0
	CEA	000111	GEN	PCTLJ	SR	-	-	Compute Effective Address
	OGT	001314	GEN	BRAN	V	6	5	Computed GOTO
	CHS	140024	GEN	INT	SRV	-	-	Change Sign
	CLS	11 03	MR	LOGIC	V	1	1	Compare L and Skip
	CMA	140401	GEN	LOGIC	SRV	-	-	Complement A
	CRA	140040	GEN	CLEAR	SRV	-	-	Clear A to 0
	CRB	140015	GEN	CLEAR	SRV	-	-	Clear B to 0
	CRE	141404	GEN	CLEAR	V	-	-	Clear E to 0
	CRL	140010	GEN	CLEAR	SRV	-	-	Clear L to 0
	CRLE	141410	GEN	CLEAR	V	-	-	Clear L and E to 0
	CSA	140320	GEN	MOVE	SRV	5	-	Copy Sign of A
	DAD	06	MR	INT	SR	2	1	Double Add
	DEL	000007	GEN	INT	SR	-	-	Enter Double Precision Mode
	DFAD	06 02	MR	FLPT	RV	3	5	Double Precision Floating Add
	DFCM	140574	GEN	FLPT	RV	3	5	Double Precision Floating Complement
	DFCS	11 02	MR	FLPT	RV	6	5	Double Precision Floating Compare and Skip

Table C-1 (continued)  
S, R, V Mode Instruction Summary

R	Mnem	Opcode	Form	Func	M	C	CC	Description
	DFDV	17 02	MR	FLPT	RV	3	5	Double Precision Floating Divide
	DFLD	02 02	MR	FLPT	RV	-	-	Double Precision Floating Load
	DFLX	15 02	MR	FLPT	V	-	-	Double Precision Floating Load Index
	DFMP	16 02	MR	FLPT	RV	3	5	Double Precision Floating Multiply
	DFSB	07 02	MR	FLPT	RV	3	5	Double Precision Floating Subtract
	DFST	04 02	MR	FLPT	RV	-	-	Double Precision Floating Store
	DIV	17	MR	INT	V	3	5	Divide
	DIV	17	MR	INT	SR	3	5	Divide
	DLD	02	MR	MOVE	SR	-	-	Double Load
	DRN	040300	GEN	FLPT	V	3	5	Double Round From Quad
	DRNM	140571	GEN	FLPT	V	8	5	Double Round From Quad Towards Negative Infinity
	DRNP	040301	GEN	FLPT	V	3	5	Double Round From Quad Towards Positive Infinity
	DRNZ	040302	GEN	FLPT	V	3	5	Double Round From Quad Towards Zero
	DRX	140210	GEN	SKIP	SRV	-	-	Decrement and Replace X
	DSB	07	MR	INT	SR	2	1	Double Subtract
	DST	04	MR	MOVE	SR	-	-	Double Store
	DVL	17 03	MR	INT	V	3	5	Divide Long
	E16S	000011	GEN	ADMOD	SRV	-	-	Enter 16S Mode
	E32I	001010	GEN	ADMOD	SRV	-	-	Enter 32I Mode
	E32R	001013	GEN	ADMOD	SRV	-	-	Enter 32R Mode
	E32S	000013	GEN	ADMOD	SRV	-	-	Enter 32S Mode
	E64R	001011	GEN	ADMOD	SRV	-	-	Enter 64R Mode
	E64V	000010	GEN	ADMOD	SRV	-	-	Enter 64V Mode
	EAA	01 01	MR	MOVE	R	-	-	Effective Address to A
	EAFA 0	001300	AP	FIELD	V	-	-	Effective Address to FAR 0
	EAFA 1	001310	AP	FIELD	V	-	-	Effective Address to FAR 1
	EAL	01 01	MR	PCTLJ	V	-	-	Effective Address to L
	EALB	13 02	MR	PCTLJ	V	-	-	Effective Address to LB
	EAXB	12 02	MR	PCTLJ	V	-	-	Effective Address to XB
R	EIO	14 01	MR	IO	V	-	7	Execute I/O
R	ENB	000401	GEN	IO	SRV	-	-	Enable Interrupts
R	ENBL	000401	GEN	IO	SRV	-	-	Enable Interrupts (Local)
R	ENBM	000400	GEN	IO	SRV	-	-	Enable Interrupts (Mutual)
R	ENBP	000402	GEN	IO	SRV	-	-	Enable Interrupts (Process)
	ERA	05	MR	LOGIC	SRV	-	-	Exclusive OR to A
	ERL	05 03	MR	LOGIC	V	-	-	Exclusive OR to L
	FAD	06 01	MR	FLPT	RV	3	5	Floating Add

Table C-1 (continued)  
S, R, V Mode Instruction Summary

R	Mnem	Opcode	Form	Func	M	C	CC	Description
	FCDQ	140571	GEN	FLPT	V	-	-	Floating Convert Double to Quad
	FCM	140530	GEN	FLPT	RV	3	5	Floating Complement
	FCS	11 01	MR	FLPT	RV	6	5	Floating Compare and Skip
	FDBL	140016	GEN	FLPT	V	-	-	Floating Convert Single to Double
	FDV	17 01	MR	FLPT	RV	3	5	Floating Divide
	FLD	02 01	MR	FLPT	RV	-	-	Floating Load
	FLOT	140550	GEN	FLPT	R	6	5	Convert Integer to Floating Point
	FLTA	140532	GEN	FLPT	V	6	5	Convert Integer to Floating Point
	FLTL	140535	GEN	FLPT	V	6	5	Convert Long Integer to Floating Point
	FLX	15 01	MR	FLPT	RV	-	-	Floating Load Index
	FMP	16 01	MR	FLPT	RV	3	5	Floating Multiply
	FRN	140534	GEN	FLPT	RV	3	5	Floating Round
	FRNM	040320	GEN	FLPT	V	3	5	Floating Round Towards Negative Infinity
	FRNP	040303	GEN	FLPT	V	3	5	Floating Round Towards Positive Infinity
	FRNZ	040321	GEN	FLPT	V	3	5	Floating Round Towards Zero
	FSB	07 01	MR	FLPT	RV	3	5	Floating Subtract
	FSGT	140515	GEN	FLPT	RV	-	5	Floating Skip If Greater Than 0
	FSLE	140514	GEN	FLPT	RV	-	5	Floating Skip If Less Than or Equal to 0
	FSMI	140512	GEN	FLPT	RV	-	5	Floating Skip If Minus
	FSNZ	140511	GEN	FLPT	RV	-	5	Floating Skip If Not Equal to 0
	FSPL	140513	GEN	FLPT	RV	-	5	Floating Skip If Plus
	FST	04 01	MR	FLPT	RV	3	5	Floating Store
	FSZE	140510	GEN	FLPT	RV	-	5	Floating Skip If Equal to 0
R	HLT	000000	GEN	MCTL	SRV	-	-	Halt
	IAB	000201	GEN	MOVE	SRV	-	-	Interchange A and B
	ICA	141340	GEN	MOVE	SRV	-	-	Interchange Bytes of A
	ICL	141140	GEN	MOVE	SRV	-	-	Interchange Bytes and Clear Left
	ICR	141240	GEN	MOVE	SRV	-	-	Interchange Bytes and Clear Right
	ILE	141414	GEN	MOVE	V	-	-	Interchange L and E
	IMA	13	MR	MOVE	SRV	-	-	Interchange Memory and A
R	INA	54	PIO	IO	SR	-	-	Input to A
R	INBC	001217	AP	PRCEX	V	6	5	Interrupt Notify Beginning, Clear Active Interrupt
R	INEN	001215	AP	PRCEX	V	6	5	Interrupt Notify Beginning

Table C-1 (continued)  
S, R, V Mode Instruction Summary

R	Mnem	Opcode	Form	Func	M	C	CC	Description
R	INEC	001216	AP	PRCEX	V	6	5	Interrupt Notify End, Clear Active Interrupt
R	INEN	001214	AP	PRCEX	V	6	5	Interrupt Notify End
R	INH	001001	GEN	IO	SRV	-	-	Inhibit Interrupts
R	INHL	001001	GEN	IO	SRV	-	-	Inhibit Interrupts (Local)
R	INHM	001000	GEN	IO	SRV	-	-	Inhibit Interrupts (Mutual)
R	INHP	001002	GEN	IO	SRV	-	-	Inhibit Interrupts (Process)
	INK	000043	GEN	KEYS	SR	-	-	Input Keys
	INT	140554	GEN	FLPT	R	3	5	Convert Floating Point to Integer
	INTA	140531	GEN	FLPT	V	3	5	Convert Floating Point to Integer
	INTL	140533	GEN	FLPT	V	3	5	Convert Floating Point to Integer Long
	IRS	12	MR	SKIP	SRV	-	-	Increment and Replace Memory
R	IRTC	000603	GEN	IO	V	7	6	Interrupt Return, Clear Active Interrupt
R	IRTN	000601	GEN	IO	V	7	6	Interrupt Return
	IRX	140114	GEN	SKIP	SRV	-	-	Increment and Replace X
R	ITLB	000615	GEN	MCTL	V	6	5	Invalidate STLB Entry
	JDX	15 02	MR	PCTLJ	R	-	-	Jump and Decrement X
	JIX	15 03	MR	PCTLJ	R	-	-	Jump and Increment X
	JMP	01	MR	PCTLJ	SRV	-	-	Jump
	JST	10	MR	PCTLJ	SRV	-	-	Jump and Store
	JSX	35 03	MR	PCTLJ	RV	-	-	Jump and Save in X
	JSXB	14 02	MR	PCTLJ	V	-	-	Jump and Save in XB
	JSY	14	MR	PCTLJ	V	-	-	Jump and Save in Y
	LCEQ	141503	GEN	LTSTS	V	-	-	Load A on Condition Code EQ
	LOGE	141504	GEN	LTSTS	V	-	-	Load A on Condition Code GE
	LOGT	141505	GEN	LTSTS	V	-	-	Load A on Condition Code GT
	LGLE	141501	GEN	LTSTS	V	-	-	Load A on Condition Code LE
	LCLT	141500	GEN	LTSTS	V	-	-	Load A on Condition Code LT
	LCNE	141502	GEN	LTSTS	V	-	-	Load A on Condition Code NE
	LDA	02	MR	MOVE	SRV	-	-	Load A
	LDC 0	001302	CHAR	CHAR	V	-	7	Load Character
	LDC 1	001312	CHAR	CHAR	V	-	7	Load Character
	LDL	02 03	MR	MOVE	V	-	-	Load Long
P	LDLR	05 01	MR	MOVE	V	-	5	Load from Addressed Register
	LDX	35 00	MR	MOVE	SRV	-	-	Load X
	LDY	35 01	MR	MOVE	V	-	-	Load Y
	LEQ	140413	GEN	LTSTS	SRV	-	4	Load A on A Equal to 0
	LF	140416	GEN	LTSTS	SRV	-	5	Load False
	LFEQ	141113	GEN	LTSTS	V	-	4	Load A on F Equal to 0
	LFGE	141114	GEN	LTSTS	V	-	4	Load A on F Greater Than or Equal to 0
	LFGT	141115	GEN	LTSTS	V	-	4	Load A on F Greater Than 0

Table C-1 (continued)  
S, R, V Mode Instruction Summary

R	Mnem	Opcode	Form	Func	M	C	CC	Description
	LFILE	141111	GEN	LTSTS	V	-	4	Load A on F Less Than or Equal to 0
	LFLI 0	001303	BRAN	FIELD	V	-	-	Load FLR 0 Immediate
	LFLI 1	001313	BRAN	FIELD	V	-	-	Load FLR 1 Immediate
	LFLT	141110	GEN	LTSTS	V	-	4	Load A on F Less Than 0
	LFNE	141112	GEN	LTSTS	V	-	4	Load A on F Not Equal to 0
	LGE	140414	GEN	LTSTS	SRV	-	4	Load A on A Greater Than or Equal to 0
	LGT	140415	GEN	LTSTS	SRV	-	4	Load A on A Greater Than 0
R	LIOT	000044	AP	MCIL	V	6	5	Load IOTLB
	LLE	140411	GEN	LTSTS	SRV	-	4	Load A on A Less Than or Equal to 0
	LLEQ	141513	GEN	LTSTS	V	-	4	Load L on A Equal to 0
	LLGE	140414	GEN	LTSTS	V	-	4	Load L on A Greater Than or Equal to 0
	LLGT	141515	GEN	LTSTS	V	-	4	Load L on A Greater Than 0
	LLL	0410XX	SHFT	SHIFT	SRV	4	-	Long Left Logical
	LLLE	141511	GEN	LTSTS	V	-	4	Load L on A Less Than or Equal to 0
	LLLT	140410	GEN	LTSTS	V	-	4	Load L on A Less Than 0
	LLNE	141512	GEN	LTSTS	V	-	4	Load L on A Not Equal to 0
	LLR	0412XX	SHFT	SHIFT	SRV	4	-	Long Left Rotate
	LLS	0411XX	SHFT	SHIFT	SRV	3	5	Long Left Shift
	LLT	140410	GEN	LTSTS	SRV	-	4	Load A on A Less Than 0
	LNE	140412	GEN	LTSTS	SRV	-	4	Load A on A Not Equal to 0
R	LPID	000617	GEN	MCIL	V	-	-	Load Process ID
R	LPSW	000711	AP	MCIL	V	7	6	Load Process Status Word
	LRL	0400XX	SHFT	SHIFT	SRV	4	-	Long Right Logical
	LRR	0402XX	SHFT	SHIFT	SRV	4	-	Long Right Rotate
	LRS	0401XX	SHFT	SHIFT	SRV	4	-	Long Right Shift
	LT	140417	GEN	LTSTS	SRV	-	5	Load True
	MPL	16 03	MR	INT	V	*	-	Multiply Long
	MPY	16	MR	INT	V	3	-	Multiply
	MPY	16	MR	INT	SR	3	*	Multiply
R	NFYB	001211	AP	PRCEX	V	6	5	Notify
R	NFYE	001210	AP	PRCEX	V	6	5	Notify
	NOP	000001	GEN	MCIL	SRV	-	-	No Operation
R	OCF	14	PIO	IO	SR	-	-	Output Control Pulse
	ORA	03 02	MR	LOGIC	V	-	-	Inclusive OR
R	OTA	74	PIO	IO	SR	-	-	Output from A
	OTK	000405	GEN	KEYS	SR	7	6	Output Keys
	PCL	10 02	MR	PCILJ	V	6	5	Procedure Call
	PID	000211	GEN	INT	SR	-	-	Position for Integer Divide
	PIDA	000115	GEN	INT	V	-	-	Position for Integer Divide
	PIDL	000305	GEN	INT	V	-	-	Position for Integer Divide Long
	PIM	000205	GEN	INT	SR	-	-	Position after Multiply

Table C-1 (continued)  
S, R, V Mode Instruction Summary

R	Mnem	Opcode	Form	Func	M	C	CC	Description
	PIMA	000015	GEN	INT	V	3	5	Position after Multiply
	PIML	000301	GEN	INT	V	3	5	Position after Multiply Long
	PRTN	000611	GEN	PCILJ	V	7	6	Procedure Return
R	PTLB	000064	GEN	MCTL	V	6	5	Purge TLB
	QFAD	5 2 2	MR	FLPT	V	3	5	Quad Precision Floating Add
	QFCM	140570	GEN	FLPT	V	3	5	Quad Precision Floating Complement
	QFCS	5 2 6	MR	FLPT	V	6	5	Quad Precision Floating Compare and Skip
	QFDV	5 2 5	MR	FLPT	V	3	5	Quad Precision Floating Divide
	QFLD	5 2 0	MR	FLPT	V	-	-	Quad Precision Floating Load
	QFLX	6 7	MR	FLPT	V	-	-	Quad Precision Floating Load Index
	QFMP	5 2 4	MR	FLPT	V	3	5	Quad Precision Floating Multiply
	QFSB	5 2 3	MR	FLPT	V	3	5	Quad Precision Floating Subtract
	QFST	5 2 1	MR	FLPT	V	-	-	Quad Precision Floating Store
	QINQ	140572	GEN	FLPT	V	3	5	Quad to Integer, in Quad Convert
	QIQR	140573	GEN	FLPT	V	3	5	Quad to Integer, in Quad Convert Rounded
	RBQ	141715	AP	QUEUE	V	-	7	Remove Entry from Bottom of Queue
	RCB	140200	GEN	KEYS	SRV	8	-	Reset CBIT to 0
R	RMC	000021	GEN	INTGY	SRV	-	-	Reset Machine Check Flag to 0
	RRST	000717	AP	MCTL	V	-	-	Restore Registers
	RSBV	000715	AP	MCTL	V	-	-	Save Registers
	RTQ	141714	AP	QUEUE	V	-	7	Remove Entry from Top of Queue
R	RTS	000511	GEN	MCTL	V	-	-	Reset Time Slice
	S1A	140110	GEN	INT	SRV	2	1	Subtract 1 from A
	S2A	140310	GEN	INT	SRV	2	1	Subtract 2 from A
	SAR	10026X	GEN	SKIP	SRV	-	-	Skip on A Register Bit Reset to 0
	SAS	10126X	GEN	SKIP	SRV	-	-	Skip on A Register Bit Set to 1
	SBL	07 03	MR	INT	V	2	1	Subtract Long
	SCB	140600	GEN	KEYS	SRV	5	-	Set CBIT to 1
	SGL	000005	GEN	INT	SR	-	-	Enter Single Precision Mode
	SGT	100220	GEN	SKIP	SRV	-	-	Skip on A Greater Than 0
	SKP	100000	GEN	SKIP	SRV	-	-	Skip
R	SKS	34	PIO	IO	SR	-	-	Skip on Condition Satisfied

Table C-1 (continued)  
S, R, V Mode Instruction Summary

R	Mnem	Opcode	Form	Func	M	C	CC	Description
	SLE	101220	GEN	SKIP	SRV	-	-	Skip on A Less Than or Equal to 0
	SLN	101100	GEN	SKIP	SRV	-	-	Skip on LSB of A Nonzero
	SLZ	100100	GEN	SKIP	SRV	-	-	Skip on LSB of A Zero
	SMCR	100200	GEN	INTGY	SRV	-	-	Skip on Machine Check Reset to 0
	SMCS	101200	GEN	INTGY	SRV	-	-	Skip on Machine Check Set to 1
	SMI	101400	GEN	SKIP	SRV	-	-	Skip on A Minus
	SNZ	101040	GEN	SKIP	SRV	-	-	Skip on A Nonzero
	SPL	100400	GEN	SKIP	SRV	-	-	Skip on A Plus
	SRC	100001	GEN	SKIP	SRV	-	-	Skip on CBIT Reset to 0
	SSC	101001	GEN	SKIP	SRV	-	-	Skip on CBIT Set to 1
	SSM	140500	GEN	INT	SRV	-	-	Set Sign of A Minus
	SSP	140100	GEN	INT	SRV	-	-	Set Sign of A Plus
	SSSN	040310	GEN	MCTL	V	6	5	Store System Serial Number
	STA	04	MR	MOVE	SRV	-	-	Store A into Memory
	STAC	001200	AP	MOVE	V	-	7	Store A Conditionally
	STC 0	001322	CHAR	CHAR	V	-	7	Store Character
	STC 1	001332	CHAR	CHAR	V	-	7	Store Character
	STEX	001315	GEN	PCTLJ	V	6	5	Stack Extend
	STFA 0	001320	AP	FIELD	V	-	-	Store FAR 0
	STFA 1	001330	AP	FIELD	V	-	-	Store FAR 1
	STL	04 03	MR	MOVE	V	-	-	Store Long
	STLC	001204	AP	MOVE	V	-	7	Store L Conditionally
P	STLR	03 01	MR	MOVE	V	-	5	Store L into Addressed Register
R	STPM	000024	GEN	MCTL	V	-	-	Store Processor Model Number
	STTM	000510	GEN	MCTL	V	6	5	Store Process Timer
	STX	15	MR	MOVE	SRV	-	-	Store X
	STY	35 02	MR	MOVE	V	-	-	Store Y
	SUB	07	MR	INT	SRV	2	1	Subtract
	SVC	000505	GEN	PCTLJ	SRV	-	-	Supervisor Call
	SZE	100040	GEN	SKIP	SRV	-	-	Skip on A Zero
	TAB	140314	GEN	MOVE	V	-	-	Transfer A to B
	TAK	001015	GEN	KEYS	V	7	6	Transfer A to Keys
	TAX	140504	GEN	MOVE	V	-	-	Transfer A to X
	TAY	140505	GEN	MOVE	V	-	-	Transfer A to Y
	TBA	140604	GEN	MOVE	V	-	-	Transfer B to A
	TCA	140407	GEN	INT	SRV	2	1	Two's Complement A
	TCL	141210	GEN	INT	V	2	1	Two's Complement Long
	TFL 0	001323	GEN	FIELD	V	-	-	Transfer FLR 0 to L
	TFL 1	001333	GEN	FIELD	V	-	-	Transfer FLR 1 to L
	TKA	001005	GEN	KEYS	V	-	-	Transfer Keys to A
	TLFL 0	001321	GEN	FIELD	V	-	-	Transfer L to FLR 0
	TLFL 1	001331	GEN	FIELD	V	-	-	Transfer L to FLR 1
	TSTQ	141757	AP	QUEUE	V	-	7	Test Queue

Table C-1 (continued)  
S, R, V Mode Instruction Summary

R	Mnem	Opcode	Form	Func	M	C	CC	Description
	TXA	141034	GEN	MOVE	V	-	-	Transfer X to A
	TYA	141124	GEN	MOVE	V	-	-	Transfer Y to A
R	WAIT	000315	AP	PRCEX	V	-	-	Wait
	XAD	001100	DECI	DECI	V	3	1	Decimal Add
	XBD	001145	DECI	DECI	V	3	5	Binary to Decimal Conversion
	XCA	140104	GEN	MOVE	SRV	-	-	Exchange and Clear A
	XCB	140204	GEN	MOVE	SRV	-	-	Exchange and Clear B
	XCM	001102	DECI	DECI	V	-	1	Decimal Compare
	XDTB	001146	DECI	DECI	V	3	5	Decimal to Binary Conversion
	XDV	001107	DECI	DECI	V	3	5	Decimal Divide
	XEC	01 02	MR	PCTLJ	RV	-	-	Execute
	XED	001112	DECI	DECI	V	-	-	Numeric Edit
	XMP	001104	DECI	DECI	V	3	1	Decimal Multiply
	XMV	001101	DECI	DECI	V	3	1	Decimal Move
	ZCM	001117	CHAR	CHAR	V	6	7	Compare Character Field
	ZED	001111	CHAR	CHAR	V	-	-	Character Field Edit
	ZFIL	001116	CHAR	CHAR	V	6	5	Fill Field With Character
	ZMV	001114	CHAR	CHAR	V	6	5	Move Character Field
	ZMVD	001115	CHAR	CHAR	V	6	5	Move Characters Between Equal Length Strings
	ZTRN	001110	CHAR	CHAR	V	-	-	Character String Translate

Table C-2  
I Mode Instruction Summary

R	Mnem	Opcode	RI	Form	Func	C	CC	Description
	A	02	RI	MRGR	INT	2	1	Add Fullword
	ABQ	134		AP	QUEUE	-	7	Add Entry to Bottom of Queue
	ACP	55	RI	GR	CPTR	-	-	Add C Pointer
	ADLR	014		RGEN	INT	2	1	Add LINK to R
	AH	12	RI	MRGR	INT	2	1	Add Halfword
	AIP	75		MRGR	GRR	2	1	Add Indirect Pointer
	ARFA 0	161		RGEN	FIELD	-	-	Add R to FAR 0
	ARFA 1	171		RGEN	FIELD	-	-	Add R to FAR 1
	ARGT	000605		GEN	PCILJ	6	5	Argument Transfer
	ATQ	135		AP	QUEUE	-	7	Add Entry to Top of Queue
	BCEQ	141602		BRAN	BRAN	-	-	Branch on Condition Code EQ
	BCGE	141605		BRAN	BRAN	-	-	Branch on Condition Code GE
	BCGT	141601		BRAN	BRAN	-	-	Branch on Condition Code GT
	BCLE	141600		BRAN	BRAN	-	-	Branch on Condition Code LE
	BCLT	141604		BRAN	BRAN	-	-	Branch on Condition Code LT
	BCNE	141603		BRAN	BRAN	-	-	Branch on Condition Code NE
	BCR	141705		BRAN	BRAN	-	-	Branch on CBIT Reset to 0
	BCS	141704		BRAN	BRAN	-	-	Branch on CBIT Set to 1
	BFEQ	122		IBRN	BRAN	-	4	Branch on F Equal to 0
	BFGE	125		IBRN	BRAN	-	4	Branch on F Greater Than or Equal to 0
	BFGT	121		IBRN	BRAN	-	4	Branch on F Greater Than 0
	BFLE	120		IBRN	BRAN	-	4	Branch on F Less Than or Equal to 0
	BFLT	124		IBRN	BRAN	-	4	Branch on F Less Than 0
	BFNE	123		IBRN	BRAN	-	4	Branch on F Not Equal to 0
	BHD1	144		IBRN	BRAN	-	-	Branch on r Decrementd by 1
	BHD2	145		IBRN	BRAN	-	-	Branch on r Decrementd by 2
	BHD4	146		IBRA	BRAN	-	-	Branch on r Decrementd by 4
	BHEQ	112		IBRN	BRAN	-	4	Branch on r Equal to 0
	BHGE	115		IBRN	BRAN	-	4	Branch on r Greater Than or Equal to 0
	BHGT	111		IBRN	BRAN	-	4	Branch on r Greater Than 0
	BHI1	140		IBRN	BRAN	-	-	Branch on r Incremented by 1
	BHI2	141		IBRN	BRAN	-	-	Branch on r Incremented by 2
	BHI4	142		IBRN	BRAN	-	-	Branch on r Incremented by 4
	BHLE	110		IBRN	BRAN	-	4	Branch on r Less Than or Equal to 0
	BHLT	114		IBRN	BRAN	-	4	Branch on r Less Than 0
	BHNE	113		IBRN	BRAN	-	4	Branch on r Not Equal to 0
	BLR	141707		BRAN	BRAN	-	-	Branch on LINK Reset to 0
	ELS	141706		BRAN	BRAN	-	-	Branch on LINK Set to 1
	BMEQ	141602		BRAN	BRAN	-	-	Branch on Magnitude Condition EQ
	BMGE	141706		BRAN	BRAN	-	-	Branch on Magnitude Condition GE

Table C-2 (continued)  
I Mode Instruction Summary

R	Mnem	Opcode	RI	Form	Func	C	CC	Description
	BMGT	141710		BRAN	BRAN	-	-	Branch on Magnitude Condition GT
	BMLE	141711		BRAN	BRAN	-	-	Branch on Magnitude Condition LE
	BMLT	141707		BRAN	BRAN	-	-	Branch on Magnitude Condition LT
	BMNE	141603		BRAN	BRAN	-	-	Branch on Magnitude Condition NE
	BRBR	040-077		IBRN	BRAN	-	-	Branch on Register Bit Reset to 0
	BRBS	000-037		IBRN	BRAN	-	-	Branch on Register Bit Set to 1
	BRD1	134		IBRN	BRAN	-	-	Branch on R Decrementd by 1
	BRD2	135		IBRN	BRAN	-	-	Branch on R Decrementd by 1
	BRD4	136		IBRN	BRAN	-	-	Branch on R Decrementd by 4
	BREQ	102		IBRN	BRAN	-	4	Branch on R Equal to 0
	BRGE	105		IBRN	BRAN	-	4	Branch on R Greater Than or Equal to 0
	BRGT	101		IBRN	BRAN	-	4	Branch on R Greater Than 0
	BRI1	130		IBRN	BRAN	-	-	Branch on R Incremented by 1
	BRI2	131		IBRN	BRAN	-	-	Branch on R Incremented by 2
	BRI4	132		IBRN	BRAN	-	-	Branch on R Incremented by 4
	BRLE	100		IBRN	BRAN	-	4	Branch on R Less Than or Equal to 0
	BRLT	104		IBRN	BRAN	-	4	Branch on R Less Than 0
	BRNE	103		IBRN	BRAN	-	4	Branch on R Not Equal to 0
C		61	RI	MRGR	INT	1	1	Compare Fullword
	CALF	000705		AP	PCTLJ	6	5	Call Fault Handler
	OCF	45	R	GR	CPTR	-	1	Compare C Pointer
	OGT	026		RGEN	BRAN	6	5	Computed GOTO
	CH	71	RI	MRGR	INT	1	1	Compare Halfword
	CHS	040		RGEN	INT	-	-	Change Sign
	CMH	045		RGEN	LOGIC	-	-	Complement r
	CMR	044		RGEN	LOGIC	-	-	Complement R
	CR	056		RGEN	CLEAR	-	-	Clear R to 0
	CRBL	062		RGEN	CLEAR	-	-	Clear R High Byte 1 Right
	CRER	063		RGEN	CLEAR	-	-	Clear R High Byte 2 Right
	CRHL	054		RGEN	CLEAR	-	-	Clear R Left Halfword
	CRHR	055		RGEN	CLEAR	-	-	Clear R Right Halfword
	CSR	041		RGEN	MOVE	5	-	Copy Sign of R
D		62	RI	MRGR	INT	3	5	Divide Fullword
	DBLE	106		RGEN	FLPT	-	-	Convert Single to Double Precision Floating
	DCP	160		RGEN	CPTR	-	-	Decrement C Pointer
	DFA	15,17	RI	MRFR	FLPT	3	5	Double Precision Floating Add
	DFC	05,07	RI	MRFR	FLPT	-	1	Double Precision Floating Compare

## INSTRUCTION SUMMARY CHARTS

Table C-2 (continued)  
I Mode Instruction Summary

R	Mnem	Opcode	RI	Form	Func	C	CC	Description
	DFCM	144		RGEN	FLPT	3	5	Double Precision Floating Complement
	DFD	31,33	RI	MRFR	FLPT	3	5	Double Precision Floating Divide
	DFL	01,03	RI	MRFR	FLPT	-	-	Double Precision Floating Load
	DFM	25,27	RI	MRFR	FLPT	3	5	Double Precision Floating Multiply
	DFS	21,23	RI	MRFR	FLPT	3	5	Double Precision Floating Subtract
	DFST	11,13		MRFR	FLPT	-	-	Double Precision Floating Store
	DH	72	RI	MRGR	INT	3	5	Divide Halfword
	DH1	130		RGEN	INT	2	1	Decrement r by 1
	DH2	131		RGEN	INT	2	1	Decrement r by 2
	DM	60		MRNR	INT	-	1	Decrement Memory Fullword
	DMH	70		MRNR	INT	-	1	Decrement Memory Halfword
	DR1	124		RGEN	INT	2	1	Decrement R by 1
	DR2	125		RGEN	INT	2	1	Decrement R by 2
	DRN	040300		GEN	FLPT	3	5	Double Round From Quad
	DRNM	140571		GEN	FLPT	8	5	Double Round From Quad Towards Negative Infinity
	DRNP	040301		GEN	FLPT	3	5	Double Round From Quad Towards Positive Infinity
	DRNZ	040302		GEN	FLPT	3	5	Double Round From Quad Towards Zero
	E16S	000011		GEN	ADMOD	-	-	Enter 16S Mode
	E32I	001010		GEN	ADMOD	-	-	Enter 32I Mode
	E32R	001013		GEN	ADMOD	-	-	Enter 32R Mode
	E32S	000013		GEN	ADMOD	-	-	Enter 32S Mode
	E64R	001011		GEN	ADMOD	-	-	Enter 64R Mode
	E64V	000010		GEN	ADMOD	-	-	Enter 64V Mode
	EAFA 0	001300		AP	FIELD	-	-	Effective Address to FAR 0
	EAFA 1	001310		AP	FIELD	-	-	Effective Address to FAR 1
	EALB	42		MRNR	PCTLJ	-	-	Effective Address to LB
	EAR	63		MRGR	PCTLJ	-	-	Effective Address to R
	EAXB	52		MRNR	PCTLJ	-	-	Effective Address to XB
R	EIO	34		MRGR	IO	-	7	Execute I/O
R	ENB	000401		GEN	IO	-	-	Enable Interrupts
R	ENBL	000401		GEN	IO	-	-	Enable Interrupts (Local)
R	ENEM	000400		GEN	IO	-	-	Enable Interrupts (Mutual)
R	ENBP	000402		GEN	IO	-	-	Enable Interrupts (Process)
	FA	014,16	RI	MRFR	FLPT	3	5	Floating Add
	FC	04,06	RI	MRFR	FLPT	-	1	Floating Compare
	FCDQ	140571		GEN	FLPT	-	-	Floating Convert Double to Quad
	FCM	100		RGEN	FLPT	3	5	Floating Complement

Table C-2 (continued)  
I Mode Instruction Summary

R	Mnem	Opcode	RI	Form	Func	C	OC	Description
	FD	30,32	RI	MRFR	FLPT	3	5	Floating Divide
	FL	00,02	RI	MRFR	FLPT	-	-	Floating Load
	FLT	105,11		RGEN	FLPT	6	5	Convert Integer to Floating Point
	FLTH	102,11		RGEN	FLPT	6	5	Convert Halfword Integer to Floating Point
	FM	24,26	RI	MRFR	FLPT	3	5	Floating Multiply
	FRN	107		RGEN	FLPT	3	5	Floating Round
	FRNM	146		RGEN	FLPT	3	5	Floating Round Towards Negative Infinity
	FRNP	145		RGEN	FLPT	3	5	Floating Round Towards Positive Infinity
	FRNZ	147		RGEN	FLPT	3	5	Floating Round Towards Zero
	FS	20,22	RI	MRFR	FLPT	3	5	Floating Subtract
	FST	10,12		MRFR	FLPT	3	5	Floating Store
R	HLT	000000		GEN	MCTL	-	-	Halt
	I	41	R	MRGR	MOVE	-	-	Interchange R and Memory Fullword
	ICBL	065		RGEN	MOVE	-	-	Interchange Bytes and Clear Left
	ICBR	066		RGEN	MOVE	-	-	Interchange Bytes and Clear Right
	ICHL	060		RGEN	MOVE	-	-	Interchange Halfwords and Clear Left
	ICHR	061		RGEN	MOVE	-	-	Interchange Halfwords and Clear Right
	ICP	167		RGEN	CPIR	-	-	Increment C Pointer
	IH	51	R	MRGR	MOVE	-	-	Interchange r and and Memory Halfword
	IH1	126		RGEN	INT	2	1	Increment r by 1
	IH2	127		RGEN	INT	2	1	Increment r by 2
	IM	40		MRNR	INT	-	1	Increment Memory Fullword
	IMH	50		MRNR	INT	-	1	Increment Memory Halfword
R	INBC	001217		AP	PRCEX	6	5	Interrupt Notify Beginning, Clear Active Interrupt
R	INEN	001215		AP	PRCEX	6	5	Interrupt Notify Beginning
R	INEC	001216		AP	PRCEX	6	5	Interrupt Notify End, Clear Active Interrupt
R	INEN	001214		AP	PRCEX	6	5	Interrupt Notify End
R	INH	001001		GEN	IO	-	-	Inhibit Interrupts
R	INHL	001001		GEN	IO	-	-	Inhibit Interrupts (Local)
R	INHM	001000		GEN	IO	-	-	Inhibit Interrupts (Mutual)
R	INHP	001002		GEN	IO	-	-	Inhibit Interrupts (Process)
	INK	070		RGEN	KEYS	-	-	Input Keys
	INT	103,11		RGEN	FLPT	3	5	Convert Floating Point to Integer

Table C-2 (continued)  
I Mode Instruction Summary

R	Mnem	Opcode	RI	Form	Func	C	CC	Description
	INTH	101,11		RGEN	FLPT	3	5	Convert Floating Point to Halfword Integer
	IR1	122		RGEN	INT	2	1	Increment R by 1
	IR2	123		RGEN	INT	2	1	Increment R by 2
	IRB	064		RGEN	MOVE	-	-	Interchange r Bytes
	IRH	057		RGEN	MOVE	-	-	Interchange R Halves
R	IRTC	000603		GEN	IO	7	6	Interrupt Return, Clear Active Interrupt
R	IRTN	000601		GEN	IO	7	6	Interrupt Return
R	ITLB	000615		GEN	MCIL	6	5	Invalidate STLB Entry
	JMP	51		MRNR	PCILJ	-	-	Jump
	JSR	73		MRGR	PCILJ	-	-	Jump to Subroutine
	JSXB	61		MRNR	PCILJ	-	-	Jump and Save in XB
	L	01	RI	MRGR	MOVE	-	-	Load
	LCC	45		MRGR	CPTR	-	7	Load C Character
	LCEQ	153		RGEN	LTSTS	-	-	Load r on Condition Code EQ
	LOGE	154		RGEN	LTSTS	-	-	Load r on Condition Code GE
	LOGT	155		RGEN	LTSTS	-	-	Load r on Condition Code GT
	LCLE	151		RGEN	LTSTS	-	-	Load r on Condition Code LE
	LCLT	150		RGEN	LTSTS	-	-	Load r on Condition Code LT
	LCNE	152		RGEN	LTSTS	-	-	Load r on Condition Code NE
P	LDAR	44		MRGR	MOVE	-	5	Load from Addressed Register
	LDC 0	162		RGEN	CHAR	-	7	Load Character
	LDC 1	172		RGEN	CHAR	-	7	Load Character
	LEQ	003		RGEN	LTSTS	-	4	Load r on R Equal to 0
	LF	016		RGEN	LTSTS	-	5	Load False
	LFEQ	023,03		RGEN	LTSTS	-	4	Load r on F Equal to 0
	LFGE	024,03		RGEN	LTSTS	-	4	Load r on F Greater Than or Equal to 0
	LFGT	025,03		RGEN	LTSTS	-	4	Load r on F Greater Than 0
	LFLE	021,03		RGEN	LTSTS	-	4	Load r on F Less Than or Equal to 0
	LFLI 0	001303		BRAN	FIELD	-	-	Load FLR 0 Immediate
	LFLI 1	001313		BRAN	FIELD	-	-	Load FLR 1 Immediate
	LFLT	020,03		RGEN	LTSTS	-	4	Load r on F Less Than 0
	LFNE	022,03		RGEN	LTSTS	-	4	Load r on F Not Equal to 0
	LGE	004		RGEN	LTSTS	-	4	Load r on R Greater Than or Equal to 0
	LGT	005		RGEN	LTSTS	-	4	Load r on R Greater Than 0
	LH	11	RI	MRGR	MOVE	-	-	Load Halfword
	LHEQ	013		RGEN	LTSTS	-	4	Load r on r Equal to 0
	LHGE	004		RGEN	LTSTS	-	4	Load r on r Greater Than or Equal to 0
	LHGT	015		RGEN	LTSTS	-	4	Load r on r Greater Than 0
	LHL1	04	R	MRGR	MOVE	-	-	Load Halfword Shifted Left by 1

Table C-2 (continued)  
I Mode Instruction Summary

R	Mnem	Opcode	RI	Form	Func	C	CC	Description
	LHL2	14	R	MRGR	MOVE	-	-	Load Halfword Shifted Left by 2
	LHL3	35	R	MRGR	MOVE	-	-	Load Halfword Shifted Left by 3
	LHLE	011		RGEN	LTSTS	-	4	Load r on r Less Than or Equal to 0
	LHLT	000		RGEN	LTSTS	-	4	Load r on r Less Than 0
	LHNE	012		RGEN	LTSTS	-	4	Load r on r Not Equal to 0
R	LIOT	000044		AP	MCTL	6	5	Load IOTLB
	LIP	65		MRGR	GRR	-	-	Load Indirect Pointer
	LLE	001		RGEN	LTSTS	-	4	Load r on R Less Than or Equal to 0
	LLT	000		RGEN	LTSTS	-	4	Load r on R Less Than 0
	LNE	002		RGEN	LTSTS	-	4	Load r on R Not Equal to 0
R	LPID	000617		GEN	MCTL	-	-	Load Process ID
R	LPSW	000711		AP	MCTL	7	6	Load Process Status Word
	LT	017		RGEN	LTSTS	-	5	Load True
	M	42	RI	MRGR	INT	*	-	Multiply Fullword
	MH	52	RI	MRGR	INT	3	5	Multiply Halfword
	N	03	RI	MRGR	LOGIC	-	-	AND Fullword
R	NFYB	001211		AP	PRCEX	6	5	Notify
R	NFYE	001210		AP	PRCEX	6	5	Notify
	NH	13	RI	MRGR	LOGIC	-	-	AND Halfword
	NOP	000001		GEN	MCTL	-	-	No Operation
	O	23	RI	MRGR	LOGIC	-	-	OR Fullword
	OH	33	RI	MRGR	LOGIC	-	-	OR Halfword
	OTK	071		RGEN	KEYS	7	6	Output Keys
	PCL	41		MRNR	PCTLJ	6	5	Procedure Call
	PID	052		RGEN	INT	-	-	Position for Integer Divide
	PIDH	053		RGEN	INT	-	-	Position r for Integer Divide
	PIM	050		RGEN	INT	3	5	Position after Multiply
	PIMH	051		RGEN	INT	3	5	Position r after Multiply
	PRTN	000611		GEN	PCTLJ	7	6	Procedure Return
R	PTLB	000064		GEN	MCTL	6	5	Purge TLB
	QFAD	36		MRFR	FLPT	3	5	Quad Precision Floating Add
	QFC	47	RI	MRFR	FLPT	-	7	Quad Precision Floating Compare
	QFCM	140570		GEN	FLPT	3	5	Quad Precision Floating Complement
	QFDV	46		MRFR	FLPT	3	5	Quad Precision Floating Divide
	QFLD	34		MRFR	FLPT	-	-	Quad Precision Floating Load
	QFMP	45		MRFR	FLPT	3	5	Quad Precision Floating Multiply
	QFSB	37		MRFR	FLPT	3	5	Quad Precision Floating Subtract

Table C-2 (continued)  
I Mode Instruction Summary

R	Mnem	Opcode	RI	Form	Func	C	CC	Description
	QFST	35		MRFR	FLPT	-	-	Quad Precision Floating Store
	QINQ	140572		GEN	FLPT	3	5	Quad to Integer, in Quad Convert
	QIQR	140573		GEN	FLPT	3	5	Quad to Integer, in Quad Convert Rounded
	RBQ	133		AP	QUEUE	-	7	Remove Entry from Bottom of Queue
	RCE	140200		GEN	KEYS	8	-	Reset CBIT to 0
R	RMC	000021		GEN	INIGY	-	-	Reset Machine Check Flag to 0
	ROT	24		MRGR	SHIFT	4	-	Rotate
	RRST	000717		AP	MCTL	-	-	Restore Registers
	RSV	000715		AP	MCTL	-	-	Save Registers
	RTQ	132		RGEN	QUEUE	-	7	Remove Entry from Top of Queue
R	RTS	000511		GEN	MCTL	-	-	Reset Time Slice
	S	22	RI	MRGR	INT	2	1	Subtract Fullword
	SCB	140600		GEN	KEYS	5	-	Set CBIT to 1
	SCC	55		MRGR	CPTR	-	-	Store C Character
	SH	32	RI	MRGR	INT	2	1	Subtract Halfword
	SHA	15		MRGR	SHIFT	4	-	Shift Arithmetic
	SHL	05		MRGR	SHIFT	4	-	Shift Logical
	SHL1	076		RGEN	SHIFT	4	-	Shift R Left 1
	SHL2	077		RGEN	SHIFT	4	-	Shift R Left 2
	SHR1	120		RGEN	SHIFT	4	-	Shift R Right 1
	SHR2	121		RGEN	SHIFT	4	-	Shift R Right 2
	SL1	072		RGEN	SHIFT	4	-	Shift R Left 1
	SL2	073		RGEN	SHIFT	4	-	Shift R Left 2
	SR1	074		RGEN	SHIFT	4	-	Shift R Right 1
	SR2	075		RGEN	SHIFT	4	-	Shift R Right 2
	SSM	042		RGEN	INT	-	-	Set Sign Minus
	SSP	043		RGEN	INT	-	-	Set Sign Plus
	SSSN	040310		GEN	MCTL	6	5	Store System Serial Number
	ST	21		MRGR	MOVE	-	-	Store Fullword
P	STAR	54		MRGR	MOVE	-	5	Store into Addressed Register
	STC 0	166		RGEN	CHAR	-	7	Store Character
	STC 1	176		RGEN	CHAR	-	7	Store Character
	STCD	137		AP	MOVE	-	7	Store Conditional Fullword
	STCH	136		AP	MOVE	-	7	Store Conditional Halfword
	STEX	027		RGEN	PCTLJ	6	5	Stack Extend
	STFA 0	001320		AP	FIELD	-	-	Store FAR 0
	STFA 1	001330		AP	FIELD	-	-	Store FAR 1
	STH	31		MRGR	MOVE	-	-	Store Halfword
R	STPM	000024		GEN	MCTL	-	-	Store Processor Model Number
	STTM	000510		GEN	MCTL	6	5	Store Process Timer
	SVC	000505		GEN	PCTLJ	-	-	Supervisor Call
	TC	046		RGEN	INT	3	1	Two's Complement R
	TCH	047		RGEN	INT	3	1	Two's Complement r

Table C-2 (continued)  
I Mode Instruction Summary

R	Mnem	Opcode	RI	Form	Func	C	OC	Description
	TCNP	76	R	MRNR	CPTR	-	1	Test C Null Pointer
	TFLR 0	163		RGEN	FIELD	-	-	Transfer FLR 0 to R
	TFLR 1	173		RGEN	FIELD	-	-	Transfer FLR 1 to R
	TM	44		MRNR	MCTL	-	1	Test Memory Fullword
	TMH	54		MRNR	INT	-	1	Test Memory Halfword
	TRFL 0	165		RGEN	FIELD	-	-	Transfer R to FLR 0
	TRFL 1	175		RGEN	FIELD	-	-	Transfer R to FLR 1
	TSTQ	104		RGEN	QUEUE	-	7	Test Queue
R	WAIT	000315		AP	PRCEX	-	-	Wait
	X	43	RI	MRGR	LOGIC	-	-	Exclusive OR Fullword
	XAD	001100		DECI	DECI	3	1	Decimal Add
	XBTD	001145		DECI	DECI	3	5	Binary to Decimal Conversion
	XCM	001102		DECI	DECI	-	1	Decimal Compare
	XDTB	001146		DECI	DECI	3	5	Decimal to Binary Conversion
	XDV	001107		DECI	DECI	3	5	Decimal Divide
	XED	001112		DECI	DECI	-	-	Numeric Edit
	XH	53	RI	MRGR	LOGIC	-	-	Exclusive OR Halfword
	XMP	001104		DECI	DECI	3	1	Decimal Multiply
	XMV	001101		DECI	DECI	3	1	Decimal Move
	ZCM	001117		CHAR	CHAR	6	7	Compare Character Field
	ZED	001111		CHAR	CHAR	-	-	Character Field Edit
	ZFIL	001116		CHAR	CHAR	6	5	Fill Field With Character
	ZM	43		MRNR	CLEAR	-	-	Clear Fullword
	ZMH	53		MRNR	CLEAR	-	-	Clear Halfword
	ZMV	001114		CHAR	CHAR	6	5	Move Character Field
	ZMVD	001115		CHAR	CHAR	6	5	Move Characters Between Equal Length Strings
	ZTRN	001110		CHAR	CHAR	-	-	Character String Translate

# D

## Hardware Consideration in Performance

Several hardware considerations have bearing on performance. First, some instructions execute faster than others. To identify these, this document lists the relative instruction weights for V and I modes. Special note is made of preferred load/store, arithmetic, and bulk data move instructions for optimum execution times. Second, the type of address formation also affects execution times. To identify these, this appendix shows the relative weights of different address formations; the performance penalties for unaligned data, cache miss, STLB miss, and address traps are also shown. Recommendations are given for how to use all of this information when coding in PMA or a high level language.

Performance of emitted code or assembler coding of identified time-crucial routines requires some knowledge of instruction execution times. Prime has never given these out before for many reasons:

- Prime's 50 Series Processors are an entire line of machines that have differing performances.
- The execution time of an instruction is based on many events such as addressing mode and data alignment, making this a complex issue.
- Contractual guarantees based on published times are certain to be wrong because of the previous point.
- There is a bad correlation of instruction times to MIPs in fact, but not in the minds of the press. Hence we would mislead by giving specific times.

## INSTRUCTION SETS GUIDE

Having said that, nevertheless, tuners must tune. The following tables represent relative "best case" weights for the "perfect" 50 Series machine. No actual machine has exactly this balance, but the 6350 and 9955 II come close.

### INSTRUCTION WEIGHTS

To use these tables, locate the desired mnemonic and note its weight in units. The following abbreviations are used.

- A -- Equal to 0 if there are no PCL arguments. Equal to  $8+6*n$  where  $n$  is the number of arguments.
- D -- The number of destination digits.
- N -- In shift instructions, the number of shifts to perform. In decimal and character instructions, the number of digits or characters involved.
- S -- The number of source digits.
- # -- The number of non-zero destination digits.

# HARDWARE CONSIDERATIONS IN PERFORMANCE

Table D-1  
V Mode Instruction Weights

Mnem	Units	Mnem	Units	Mnem	Units	Mnem	Units
A1A	1	BLS	3	E64V	8	INTA	14
A2A	1	BLT	3	EAFA	3	INTL	14
ABQ	20	BMEQ	2	EAL	1	IRS	3
ACA	1	BMGE	2	EALB	2	IRTC	9
ADD	1	BMGT	3	EAXB	1	IRTN	4
ADL	1	BMGT	3	EIO	12	IRX	2
ADLL	1	BMLE	3	EMCM	4	ITLB	8
ALFA	2	BMLT	2	ENB	3	JMP	2
ALL	N+2	BMNE	2	ERA	1	JST	7
ALR	N+2	BNE	3	ERL	1	JSX	5
ALS	N+2	CAI	3	ESIM	4	JSXB	5
ANA	1	CAL	1	EVIM	4	JSY	5
ANL	1	CALF	63	FAD	4	LCEQ	2
ARGT	See PCL	CAR	1	FCM	6	LOGE	2
ARL	N+2	CAS	4	FCS	6	LOGT	2
ARR	N+2	CAZ	3	FDEL	1	LCLE	2
ARS	N+2	OGT	9	FDV	38	LCLT	2
ATQ	20	CHS	1	FLD	2	LCNE	2
BCEQ	2	CLS	4	FLTA	10	LDA	1
BOGE	2	CMA	1	FLTL	15	LDC	10
BOGT	2	CRA	1	FLX	1	LDL	1
BCLE	2	CRB	1	FMP	9	LDLR	11
BCLT	2	CRE	1	FRN	14	LDX	1
BCNE	2	CRL	1	FRNM	7	LDY	1
BCR	2	CRLE	2	FRNP	11	LEQ	3
BCS	2	CSA	1	FRNZ	12	LF	2
BDX	3	DFAD	6	FSB	4	LFEQ	4
BDY	3	DFCM	6	FSGT	4	LFGE	4
BEQ	3	DFCS	8	FSLE	4	LFGT	4
BFEQ	3	DFDV	38	FSMI	4	LFLE	4
BFGE	3	DFLD	2	FSNZ	4	LFLI	2
BFGT	3	DFLX	2	FSPL	4	LFLT	4
BFLE	3	DFMP	15	FST	3	LFNE	4
BFLT	3	DFSB	6	FSZE	4	LGE	3
BFNE	3	DFST	5	HLT	0	LGT	3
BGE	3	DIV	20	IAB	2	LIOT	37
BGT	3	DRN	14	ICA	1	LLE	3
BIX	3	DRNM	5	ICL	1	LLEQ	3
BIY	3	DRNP	10	ICR	1	LLGE	3
BLE	3	DRNZ	9	ILE	3	LLGT	3
BLEQ	3	DRX	2	IMA	4	LLL	N+2
BLGE	3	DVL	47	INBC	40	LLLE	3
BLGT	3	E16S	8	INEN	35	LLLT	3
BLLE	3	E32I	8	INEC	40	LLNE	3
BLLT	3	E32R	8	INEN	35	LLR	N+2
BLNE	3	E32S	8	INH	3	LLS	N+2
BLR	3	E64R	8	INK	6	LLT	3

Table D-1 (Continued)  
V Mode Instruction Weights

Mnem	Units	Mnem	Units	Mnem	Units	Mnem	Units
LMCM	4	QFST	19	SR3	10	TAX	1
LNE	3	QINQ	55	SR4	10	TAY	1
LPID	6	QIQR	56	SRC	3	TBA	1
LPSW	14	RBQ	20	SS1	10	TCA	2
LRL	N+2	RCB	1	SS2	10	TCL	2
LRR	N+2	RMC	15	SS3	10	TFLL	3
LRS	N+2	RRST	44	SS4	10	TKA	3
LT	2	RSBV	85	SSC	3	TLFL	2
MPL	13	RTQ	18	SSM	1	TSTQ	7
MPY	8	RTS	10	SSP	1	TXA	1
NFYB	35	S1A	1	SSR	10	TYA	1
NFYE	35	S2A	1	SSS	10	WAIT	58
NOP	1	SAR	3	SSSN	37	XAD	76+3*N
ORA	1	SAS	3	STA	2	XBTD	40+5*N
OTK	9	SBL	1	STAC	8	XCA	2
PCL	40+A	SCB	1	STC	12	XCB	2
PIDA	2	SGT	3	STEX	9	XCM	80+2*N
PIDL	3	SKP	7	STFA	8	XDTB	40+5*N
PIMA	3	SLE	3	STL	2	XDV	90+65*#
PIML	4	SLN	3	STLC	9	XEC	9
PRTN	16	SLZ	3	STLR	13	XED	Varies
PTLB	400	SMCR	5	STPM	12	XMP	88+15*S*D
QFAD	56	SMCS	5	STIM	17	XMV	80+3*N
QFCM	10	SMI	3	STX	2	ZCM	20+N
QFCS	39	SNR	10	STY	2	ZED	Varies
QFDV	489	SNS	10	SUB	1	ZFIL	14+0.5*N
QFLD	14	SNZ	3	SVC	36	ZMV	18+0.75*N
QFLX	2	SPL	3	SZE	3	ZMVD	14+0.75*N
QFMP	65	SR1	10	TAB	1	ZTRN	14+8*N
QFSB	57	SR2	10	TAK	3		

# HARDWARE CONSIDERATIONS IN PERFORMANCE

Table D-2  
I Mode Instruction Weights

Mnem	Units	Mnem	Units	Mnem	Units	Mnem	Units
A	1	BRD4	3	E16S	8	INT	14
ABQ	20	BREQ	3	E32I	8	INTH	14
ACP	4	BRGE	3	E32R	8	IR1	1
ADLR	1	BRGT	3	E32S	8	IR2	1
AH	1	BRI1	3	E64R	8	IRB	1
AIP	2	BRI2	3	E64V	8	IRH	2
ARFA	2	BRI4	3	EAFA	3	IRTC	9
ARGT	See PCL	BRLE	3	EALB	2	IRTN	4
ATQ	20	BRLT	3	EAR	1	ITLB	8
BCEQ	2	BRNE	3	EAXB	1	JMP	2
BOGE	2	C	1	EIO	12	JSR	5
BOGT	2	CAI	3	EMCM	4	JSXB	5
BCLE	2	CALF	63	ENB	3	L	1
BCLT	2	CCP	5	ESIM	4	LCC	3
BCNE	2	CGT	9	EVIM	4	LCEQ	2
BCR	2	CH	1	FA	4	LOGE	2
BCS	2	CHS	1	FC	10	LOGT	2
BFEQ	3	CMH	1	FCM	6	LCLE	2
BFGE	3	CMR	1	FD	38	LCLT	2
BFGT	3	CR	1	FL	2	LCNE	2
BFLE	3	CRBL	1	FLT	15	LDAR	11
BFLT	3	CRER	1	FLTH	10	LDC	10
BFNE	3	CRHL	1	FM	9	LEQ	3
BHD1	3	CRHR	1	FRN	14	LF	2
BHD2	3	CSR	1	FRNM	7	LFEQ	4
BHD4	3	D	47	FRNP	11	LFGE	4
BHEQ	3	DELE	1	FRNZ	12	LFGT	4
BHGE	3	DCP	3	FS	4	LFLE	4
BHGT	3	DFA	6	FST	3	LFLI	2
BHI1	3	DFC	10	HIT	0	LFLT	4
BHI2	3	DFCM	6	I	4	LFNE	4
BHI4	3	DFD	38	ICBL	1	LGE	3
BHLE	3	DFL	2	ICBR	1	LGT	3
BHLT	3	DFM	15	ICHL	1	LH	1
BHNE	3	DFS	6	ICHR	1	LHEQ	3
BLR	3	DFST	5	ICP	3	LHGE	3
BLS	3	DH	20	IH	4	LHGT	3
BMEQ	2	DH1	1	IH1	1	LHL1	1
BMGE	2	DH2	1	IH2	1	LHL2	1
BMGT	3	DM	3	IM	3	LHLE	3
BMLE	3	DMH	3	IMH	3	LHLT	3
BMLT	2	DR1	1	INBC	40	LHNE	3
BMNE	2	DR2	1	INEN	35	LIOT	37
BRER	3	DRN	14	INEC	40	LIP	2
BRBS	3	DRNM	5	INEN	35	LLE	3
BRD1	3	DRNP	10	INH	3	LLT	3
BRD2	3	DRNZ	9	INK	6	LMCM	4

Table D-2 (Continued)  
I Mode Instruction Weights

Mnem	Units	Mnem	Units	Mnem	Units	Mnem	Units
LNE	3	QFDV	489	SHR2	2	TMH	1
LPID	6	QFLD	14	SL1	1	TRFL	2
LPSW	14	QFMP	65	SL2	1	TSTQ	7
LT	2	QFSB	57	SR1	1	WAIT	58
M	13	QFST	19	SR2	2	X	1
MH	8	QINQ	55	SSM	1	XAD	76+3*N
N	1	QIQR	56	SSP	1	XBTD	40+5*N
NFYB	35	RBQ	20	SSSN	37	XCM	80+2*N
NFYE	35	RCB	1	ST	2	XDTB	40+5*N
NH	1	RMC	15	STAR	13	XDV	90+65*#
NOP	1	ROT	N+2	STC	12	XED	Varies
O	1	RRST	44	STCD	9	XH	1
OH	1	RSAB	85	STCH	9	XMP	88+15*S*D
OTK	9	RTQ	18	STEX	9	XMV	80+3*N
PCL	40+A	RTS	10	STFA	8	ZCM	20+N
PID	3	S	1	STH	2	ZED	Varies
PIDH	2	SCB	1	STPM	12	ZFIL	14+0.5*N
PIM	4	SOC	5	STTM	17	ZM	2
PIMH	3	SH	1	SVC	36	ZMH	2
PRTN	16	SHA	N+2	TC	2	ZMV	18+0.75*N
PTLB	400	SHL	N+2	TCH	2	ZMVD	14+0.75*N
QFAD	56	SHL1	1	TCNP	2	ZTRN	14+8*N
QFC	24	SHL2	1	TFLR	3		
QFCM	10	SHR1	1	TM	1		

Examination of the V and I mode instruction weights shows that certain instructions have much activity in them and thus take much longer to complete execution. Such instructions include STLR/LDLR and STAR/LDAR (both 13/11 units) that store/load the L register into the addressed register. Other such instructions are RSAB/RRST (44/85 units) that save/restore all registers.

Other instructions are very fast, such as the long loads (LDL and L) at one unit each.

Prime processor designers have worked hard to make the instructions that "feel" fast be fast. "Cute" uses of instructions are usually punished by reduced performance. An example of "cute" instruction use is LDX# 2 instead of STL Temp, LDX Temp+1. Clever use exploits the address modes and multiple index registers to save instructions. Clever use of registers can save stores, but shuffling data from one register to another (even in I mode) to save a store has little value.

Restricted instructions are shown in these tables. Even though several of these are heavily weighted, they are not discussed here since they are Ring 0 instructions.

Also, short integer (16-bit) instructions take less time to execute than long integer (32-bit) ones, particularly in the case of multiplies and divides. For V mode, long integer arithmetic mnemonics end in "L", such as MPL and DVL, while short integer ones do not, as in MPY and DIV. I mode short integer mnemonics end in "H" (half register), such as MH and DH, while long integer ones are simply M and D for multiply long and divide long.

For all processors, be sure to use the ZMVD (Move Characters between Equal Length Strings) instruction when moving bulk data. ZMVD is the most efficient means for data moving. All of Prime software is learning to use this instruction for bulk data transfers. Prime processors are optimized for ZMVD.

The advantage of using these tables of weighted instruction times is obvious if you are programming in PMA. If you are programming in a high level language such as FORTRAN or Pascal, however, you first need to generate an expanded listing when you are compiling your source program. Such a listing shows the PMA code that the compiler generated for each source statement. Simple arithmetic will show the approximate relative weights that each source statement takes.

#### EXTENSIONS TO INSTRUCTION WEIGHTS

Unfortunately, no simple calculation can accurately produce the actual instruction time of any modern machine (including all of Prime's, of course). Many factors influence the execution of a single instruction. The most important is, of course, the processor type. However, many other factors also affect execution time. Address formation and virtual memory considerations are shown in Table D-3. Other factors are harder to describe and so are deemed less important. Among them are I/O (DMx) and process exchange activity, interprocessor locking (on the P850), memory refresh, EOC's, etc.

Table D-3 shows that indexing adds no further time to the basic instruction while indirection adds 1 unit. Unaligned data also adds 1 unit, so be sure to align data on even word (32-bit) boundaries in common blocks. Prime software provides for proper data alignment if possible. Address traps add considerably more time to instruction execution. Read or write address traps add 8 units apiece, and should be avoided. An address trap is invoked in V mode short instructions if the final address is to memory from 0 to 7. The trap is to register file locations.

A time penalty is paid whenever there is a cache miss (13 units) or an STLB miss (31 units), since the virtual-to-physical address translation process has to occur. The more pages used in a program, the higher the probability of a cache miss, STLB miss or a page fault. As a rule of

thumb to keep these delays down, ensure that your programs have their most frequently-used subroutines loaded together -- do not load subroutines on an alphabetic basis.

Table D-3

Comparative Weighting  
Address Formation High End for 32I Mode

Vanilla	L	1,FOO	1 unit	
Indexed	L	1,FOO, 2	1 unit	
Reg-Reg	L	1,2	1 unit	
Immediate	L	1,=10	1 unit	
Indirect	L	1,P\$FOO,*	2 units	
Indirect	L	1,P\$FOO,*2	2 units	(postindexed)
Unaligned	L	1,FOO	2 units	
Cache miss	L	1,FOO	14 units	(1 only)
STLB miss	L	1,FOO	32 units	(1 only)
Worst case	L	1,P\$FOO,*	179 units	
(Four STLB and Cache misses, Indirect, I.P. and operand unaligned.)				
Address Traps:				
LDA#	6		9 units	(Read address trap)
DFST	TEMP		6 units	(Better practice)
LDA	TEMP+3			
STA#	6		9 units	(Write address trap)
STA	TEMP+3		4 units	(Better practice)
DFLD	TEMP			
"Normal" cache hit rate of 98 percent				
"Normal" STLB hit rate of 99 percent				

# E

## Archived Instructions

This appendix contains archived S, R, V, and I mode instructions. These instructions support options that are no longer offered, or they support functions that are no longer used. Table E-1 contains a summary of the archived instructions. (This table is in the same format as those in Appendix C.) The descriptions of these instructions follow Table E-1.

Table E-1  
Archived Instruction Summary

R	Mnem	Opcode	Form	Func	M	C	CC	Description
R	CAI	000411	GEN	IO	SRVI	-	-	Clear Active Interrupt
	CREP	02	MR	PCTLJ	R	-	-	Call Recursive Entry Procedure
	CXCS	001714	GEN	MCTL	VI	6	5	Control Extended Control Store
R	EMCM	000503	GEN	INIGY	SRVI	-	-	Enter Machine Check Mode
	ENTR	01 03	MR	PCTLJ	R	-	-	Enter Recursive Procedure Stack
R	ESIM	000415	GEN	IO	SRVI	-	-	Enter Standard Interrupt Mode
R	EVIM	000417	GEN	IO	SRVI	-	-	Enter Vectored Interrupt Mode
	JEQ	02 03	MR	PCTLJ	R	-	-	Jump on A Equal to 0
	JGE	07 03	MR	PCTLJ	R	-	-	Jump on A Greater Than or Equal to 0

Table E-1 (continued)  
Archived Instruction Summary

R	Mnem	Opcode	Form	Func	M	C	CC	Description
	JGT	05 03	MR	PCTLJ	R	-	-	Jump on A Greater Than 0
	JLE	04 03	MR	PCTLJ	R	-	-	Jump on A Less Than or Equal to 0
	JLT	06 03	MR	PCTLJ	R	-	-	Jump on A Less Than 0
	JNE	03 03	MR	PCTLJ	R	-	-	Jump on A Not Equal to 0
R	LMCM	000501	GEN	INTGY	SRVI	-	-	Leave Machine Check Mode
	LWCS	001710	GEN	MCTL	VI	6	5	Load Writable Control Store
R	MDEI	001304	GEN	INTGY	VI	6	5	Memory Diagnostic Enable Interleave
R	MDII	001305	GEN	INTGY	VI	6	5	Inhibit Interleaved
R	MDIW	001324	GEN	INTGY	VI	6	5	Write Interleaved
R	MDRS	001306	GEN	INTGY	VI	6	5	Read Syncrome Bits
R	MDWC	001307	GEN	INTGY	VI	6	5	Load Write Control Register
	MIA	64	MRGR	MCTL	I	-	-	Microcode Entrance
	MIA	12 01	MR	MCTL	V	-	-	Microcode Entrance
	MIB	74	MRGR	MCTL	I	-	-	Microcode Entrance
	MIB	13 01	MR	MCTL	V	-	-	Microcode Entrance
	NRM	000101	GEN	INT	SR	8	-	Normalize
	RTN	000105	GEN	PCTLJ	SR	-	-	Return
	SCA	000041	GEN	INT	SR	-	-	Load Shift Count into A
R	SNR	10024X	GEN	SKIP	SRV	-	-	Skip on Sense Switch N Reset to 0
R	SNS	10124X	GEN	SKIP	SRV	-	-	Skip on Sense Switch N Set to 1
R	SR1	100020	GEN	SKIP	SRV	-	-	Skip on Sense Switch 1 Reset to 0
R	SR2	100010	GEN	SKIP	SRV	-	-	Skip on Sense Switch 2 Reset to 0
R	SR3	100004	GEN	SKIP	SRV	-	-	Skip on Sense Switch 3 Reset to 0
R	SR4	100002	GEN	SKIP	SRV	-	-	Skip on Sense Switch 4 Reset to 0
R	SS1	101020	GEN	SKIP	SRV	-	-	Skip on Sense Switch 1 Set to 1
R	SS2	101010	GEN	SKIP	SRV	-	-	Skip on Sense Switch 2 Set to 1
R	SS3	101004	GEN	SKIP	SRV	-	-	Skip on Sense Switch 3 Set to 1
R	SS4	101002	GEN	SKIP	SRV	-	-	Skip on Sense Switch 4 Set to 1
R	SSR	100036	GEN	SKIP	SRV	-	-	Skip on All Sense Switches Reset to 0
R	SSS	101036	GEN	SKIP	SRV	-	-	Skip on Any Sense Switches Set to 1
	VIRY	000311	GEN	INTGY	SRVI	6	5	Verify
	WCS	0016XX	GEN	MCTL	RVI	-	-	Write Control Store
	XVRY	001113	MCTL	GEN	VI	6	5	Verify XIS

► CAI  
Clear Active Interrupt  
0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 1 (S, R, V, I mode form)

Clears the current active interrupt. Effective only in vectored interrupt mode. Inhibits interrupts for one instruction. Leaves the values of CBIT, LINK, and the condition codes unchanged.

#### Note

This is a restricted instruction.

► CREP address  
Call Recursive Entry Procedure  
I X 1 0 0 0 1 1 0 0 0 0 1 0 CB\2 (R mode form)  
[ DISPLACEMENT\16 ]

Increments the contents of the program counter and loads the result into the location following the one specified by the current value of the R mode stack pointer. Calculates an effective address, EA, and loads it into the program counter. Execution continues with the location specified by the new value of the program counter.

This instruction performs subroutine linkage for reentrant or recursive procedures. CREP stores the return address in bits 17-32 (the second halfword) of a stack frame created by the ENIR instruction, rather than in the destination address as JST does. Leaves the values of CBIT, LINK, and the condition codes indeterminate.

► CXCS  
Control Extended Control Store  
0 0 0 0 0 0 1 1 1 1 0 0 1 1 0 0 (V, I mode form)

Moves the A register contents to the control register on the writable control store board. Leaves the values of CBIT, LINK, and the condition codes indeterminate.

► EMCM  
Enter Machine Check Mode  
0 0 0 0 0 0 0 1 0 1 0 0 0 0 1 1 (S, R, V, I mode form)

Enters machine check mode 3 by loading 3 into modal bits 15-16. This mode enables the reporting of all errors. The actions taken upon an error depend on whether the machine was in process exchange mode or not.

The instruction inhibits interrupts during execution of the next instruction. Leaves the values of CBIT, LINK, and the condition codes unchanged. See Chapter 10 of the System Architecture Reference Guide for more information about checks.

If an error occurs in process exchange mode, the microcode stores the machine state in the appropriate check vector and transfers control to that vector, automatically dropping back to machine check mode 0.

If an error occurs when the machine is not in process exchange mode, the following actions occur. If the appropriate check vector contains a nonzero value, the processor jumps indirectly through this vector to the check routine. If the check vector location contains 0, the machine halts.

#### Note

This is a restricted instruction.

► ENTR n  
Enter R Mode Recursive Procedure Stack  
I X 0 0 0 1 1 1 0 0 0 0 1 1 CB\2 (R mode long form)  
[ DISPLACEMENT\16 ]

Creates a save area n halfwords long. (A halfword is 16 bits.) Saves the current value of the R mode stack pointer in the first halfword of the save area. The starting address of the save area is:

( contents of R mode stack pointer ) - n

This means that the instruction creates a stack frame containing n locations, and that the first location points to the previous frame.

The ENTR instruction leaves the values of CBIT, LINK, and the condition codes unchanged.

► ESIM  
Enter Standard Interrupt Mode  
0 0 0 0 0 0 0 1 0 0 0 0 1 1 0 1 (S, R, V, I mode form)

Enters standard interrupt mode by resetting bit 2 of the modals to 0. Inhibits interrupts for one instruction. ESIM is meaningless when the system is in process exchange mode (that is, the value of modal bit 13 is 1). All interrupts use location '63. The processor services interrupts according to their relative positions on the I/O bus. Lower devices have higher priority. Inhibits interrupts during execution of the next instruction. Leaves the values of CBIT, LINK, and the condition codes unchanged. Refer to Chapter 10 of the System Architecture Reference Guide for more information about interrupts.

Note

ESIM is a restricted instruction.

► **EVIM**  
 Enter Vectored Interrupt Mode  
 0 0 0 0 0 0 0 1 0 0 0 0 1 1 1 1 (S, R, V, I mode form)

Enters vectored interrupt mode by setting bit 2 of the modals to 1. EVIM is meaningless when the system is in process exchange mode (that is, the value of modal bit 13 is 1). The processor services interrupts according to their relative positions on the I/O bus. Lower devices have higher priority. Interrupts occur through a location specified by the interrupting device. Inhibits interrupts during execution of the next instruction. Leaves the values of LINK, CBIT, and the condition codes unchanged. Refer to Chapter 10 of the System Architecture Reference Guide for more information about interrupts.

Note

This is a restricted instruction.

► **JEQ address**  
 Jump on A Equal to 0  
 I X 0 0 1 0 1 1 0 0 0 0 1 1 CB\2 (R mode form)  
 [ DISPLACEMENT\16 ]

Calculates an effective address, EA. Loads EA into the program counter if the contents of A are equal to 0. If the contents of A are not equal to 0, execution continues with the next instruction. Leaves the values of CBIT, LINK, and the condition codes unchanged.

► **JGE address**  
 Jump on A Greater Than or Equal to 0  
 I X 0 1 1 1 1 1 0 0 0 0 1 1 CB\2 (R mode form)  
 [ DISPLACEMENT\16 ]

Calculates an effective address, EA. If the contents of A are greater than or equal to 0, the instruction loads EA into the program counter. If the contents of A are less than 0, execution continues with the next instruction. Leaves the contents of CBIT, LINK, and the condition codes unchanged.

► JGT address  
 Jump on A Greater Than 0  
 I X 0 1 0 1 1 1 0 0 0 0 1 1 CB\2 (R mode form)  
 [ DISPLACEMENT\16 ]

Calculates an effective address, EA. If the contents of A are greater than 0, the instruction loads EA into the program counter. If the contents of A are less than or equal to 0, execution continues with the next instruction. Leaves the contents of CBIT, LINK, and the condition codes unchanged.

► JLE address  
 Jump on A Less Than or Equal to 0  
 I X 0 1 0 0 1 1 0 0 0 0 1 1 CB\2 (R mode form)  
 [ DISPLACEMENT\16 ]

Calculates an effective address, EA. If the contents of A are less than or equal to 0, the instruction loads EA into the program counter. If the contents of A are greater than 0, execution continues with the next instruction. Leaves the contents of LINK, CBIT, and the condition codes unchanged.

► JLT address  
 Jump on A Less Than 0  
 I X 0 1 1 0 1 1 0 0 0 0 1 1 CB\2 (R mode form)  
 [ DISPLACEMENT\16 ]

Calculates an effective address, EA. If the contents of A are less than 0, the instruction loads EA into the program counter. If the contents of A are greater than 0, execution continues with the next instruction. Leaves the contents of CBIT, LINK, and the condition codes unchanged.

► JNE address  
 Jump on A Not Equal to 0  
 I X 0 0 1 1 1 1 0 0 0 0 1 1 CB\2 (R mode form)  
 [ DISPLACEMENT\16 ]

Calculates an effective address, EA. If the contents of A do not equal 0, the instruction loads EA into the program counter. If the contents of A are equal to 0, execution continues with the next instruction. Leaves the contents of CBIT, LINK, and the condition codes unchanged.

► **LMCM**  
 Leave Machine Check Mode  
 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 1 (S, R, V, I mode form)

Leaves machine check mode by setting bits 15-16 of the modals to 00. If a machine parity error occurs in this mode, the hardware sets the machine check flag but no check (V mode) or interrupt (S, R modes) occurs. Inhibits the machine for one instruction. Leaves the values of CBIT, LINK, and the condition codes unchanged.

Note

This is a restricted instruction.

► **LWCS**  
 Load Writable Control Store  
 0 0 0 0 0 0 1 1 1 1 0 0 1 0 0 0 (V, I mode form)

Loads the writable control store portion of the extended control store board from the memory block pointed to by XB. The control register loaded by CXCS modifies this instruction. Leaves the values of CBIT, LINK, and the condition codes indeterminate.

► **MDEI**  
 Memory Diagnostic Enable Interleave  
 0 0 0 0 0 0 1 0 1 1 0 0 0 1 0 0 (V, I mode form)

Enables the memory interleave facility. Leaves the values of LINK, CBIT, and the condition codes unchanged.

Note

This is a restricted instruction.

► **MDII**  
 Memory Diagnostic Inhibit Interleave  
 0 0 0 0 0 0 1 0 1 1 0 0 0 1 0 1 (V, I mode form)

Inhibits the memory interleave facility. Leaves the values of LINK, CBIT, and the condition codes unchanged.

Note

This is a restricted instruction.

► **MDIW**  
 Memory Diagnostic Write Interleaved  
 0 0 0 0 0 0 1 0 1 1 0 1 0 1 0 0 (V, I mode form)

Writes interleaved memory. Leaves the values of LINK, CBIT, and the condition codes unchanged.

Note

This is a restricted instruction.

► **MDRS**  
 Memory Diagnostic Read Syndrome Bits  
 0 0 0 0 0 0 1 0 1 1 0 0 0 1 1 0 (V, I mode form)

Reads memory syndrome bits. Leaves the values of LINK, CBIT, and the condition codes unchanged.

Note

This is a restricted instruction.

► **MDWC**  
 Memory Diagnostic Load Write Control Register  
 0 0 0 0 0 0 1 0 1 1 0 0 0 1 1 1 (V, I mode form)

Writes memory control register. Leaves the values of LINK, CBIT, and the condition codes unchanged.

Note

This is a restricted instruction.

► **MIA**  
 Microcode Execute A  
 I X 1 0 1 0 1 1 0 0 0 Y 0 1 BR\2 (V mode long form)  
 DISPLACEMENT\16  
  
 1 1 0 1 0 0 DR\3 TM\2 SR\3 BR\2 (I mode form)  
 [ DISPLACEMENT\16 ]

This instruction currently causes a UII fault. If implemented, this instruction is for user-written microcode. For more information about UII, refer to Chapter 10 of the System Architecture Reference Guide.

► **MIB**  
 Microcode Execute B  
 I X 1 0 1 1 1 1 0 0 0 Y 0 1 BR\2 (V mode long)  
 DISPLACEMENT\16  
  
 1 1 1 1 0 0 DR\3 TM\2 SR\3 BR\2 (I mode form)  
 [ DISPLACEMENT\16 ]

This instruction currently causes a UII fault. If implemented, this instruction is for user-written microcode. For more information about UII, refer to Chapter 10 of the System Architecture Reference Guide.

► **NRM**  
 Normalize  
 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 (S, R mode form)

Shifts the 31-bit integer in A and B to the left arithmetically, shifting in 0s into bit 16 of B. The shift does not affect bit 1 of B or bit 1 of A. The instruction shifts bits out of bit 2 in A until the value of bit 2 is opposite the value of bit 1 in A. Loads bits 9-16 of the S and R mode keys with the number of shifts performed.

Normalizing 0 on all machines results in the following: zeros are loaded in bits 9-16 of the keys; bit 1 of the B register is ignored in the test for zero. Bit 1 of the B register may be reset or left unchanged, depending on the processor.

Leaves the values of CBIT and the condition codes unchanged; the value of LINK is indeterminate.

#### Note

Since the bits shifted out of bit 2 in A contain copies of the sign of the 31-bit number, the shift results in no loss of information.

► **RTN**  
 Return  
 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 1 (R mode form)

Returns control from a P300 recursive procedure to the calling routine. To do this, RTN fetches the return address from the second halfword of the previous stack frame and loads the result in the program counter. RTN then transfers halfword 1 (the pointer to the preceding stack frame) to the S register. (A halfword is 16 bits.)

(S)+1 -> P  
 (S) -> S

If the return address is 0, (S) is unchanged and a PSU (Procedure Stack Underflow) fault is taken (interrupt through location '75 in physical memory is taken on the Prime 300). Leaves the values of LINK, CBIT, and the condition codes unchanged.

Note

This instruction reverses the actions done by CREP and ENTR.

► SCA  
Load Shift Count Into A  
0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 (S, R mode form)

Loads the contents of bits 9-16 of the keys into bits 9-16 of A. Clears bits 1-8 of A to 0. Leaves the values of CBIT, LINK, and the condition codes unchanged.

Note

The SCA instruction is used with NRM.

► SNR n  
Skip on Sense Switch N Reset to 0  
1 0 0 0 0 0 0 0 1 0 1 0 N\4 (S, R, V mode form)

Skips the next sequential 16-bit halfword if the contents of sense switch N are 0. Leaves the values of CBIT, LINK, and the condition codes unchanged.

N specifies the sense switch to test.

Note

This is a restricted instruction.

► SNS  
Skip on Sense Switch N Set to 1  
1 0 0 0 0 0 1 0 1 0 1 0 N\4 (S, R, V mode form)

Skips the next sequential 16-bit halfword if the value of sense switch N is 1. Leaves the values of CBIT, LINK, and the condition codes unchanged.

N specifies the sense switch to test.

Note

SNS is a restricted instruction.

► SR1  
Skip on Sense Switch 1 Reset to 0  
1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 (S, R, V mode form)

Skips the next sequential 16-bit halfword if the value of sense switch 1 is 0. Leaves the values of CBIT, LINK, and the condition codes unchanged.

Note

This is a restricted instruction.

► SR2  
Skip on Sense Switch 2 Reset to 0  
1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 (S, R, V mode form)

Skips the next sequential 16-bit halfword if the value of sense switch 2 is 0. Leaves the values of CBIT, LINK, and the condition codes unchanged.

Note

This is a restricted instruction.

► SR3  
Skip on Sense Switch 3 Reset to 0  
1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 (S, R, V mode form)

Skips the next sequential 16-bit halfword if the value of sense switch 3 is 0. Leaves the values of CBIT, LINK, and the condition codes unchanged.

Note

This is a restricted instruction.

- **SR4**  
 Skip on Sense Switch 4 Reset to 0  
 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 (S, R, V mode form)

Skips the next sequential 16-bit halfword if the value of sense switch 4 is 0. Leaves the values of CBIT, LINK, and the condition codes unchanged.

Note

This is a restricted instruction.

- **SS1**  
 Skip on Sense Switch 1 Set to 1  
 1 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 (S, R, V mode form)

Skips the next sequential 16-bit halfword if the value of sense switch 1 is 1. Leaves the values of CBIT, LINK, and the condition codes unchanged.

Note

This is a restricted instruction.

- **SS2**  
 Skip on Sense Switch 2 Set to 1  
 1 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 (S, R, V mode form)

Skips the next sequential 16-bit halfword if the value of sense switch 2 is 1. Leaves the values of CBIT, LINK, and the condition codes unchanged.

Note

This is a restricted instruction.

- **SS3**  
 Skip on Sense Switch 3 Set to 1  
 1 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 (S, R, V mode form)

Skips the next sequential 16-bit halfword if the value of sense switch 3 is 1. Leaves the values of CBIT, LINK, and the condition codes unchanged.

Note

This is a restricted instruction.

► SS4  
Skip on Sense Switch 4 Set to 1  
1 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 (S, R, V mode form)

Skips the next sequential 16-bit halfword if the value of sense switch 4 is 1. Leaves the values of CBIT, LINK, and the condition codes unchanged.

Note

This is a restricted instruction.

► SSR  
Skip on All Sense Switches Reset to 0  
1 0 0 0 0 0 0 0 0 0 0 1 1 1 1 0 (S, R, V mode form)

Skips the next sequential 16-bit halfword if the values of sense switches 1, 2, 3, and 4 are all 0. Leaves the values of CBIT, LINK, and the condition codes unchanged.

Note

This is a restricted instruction.

► SSS  
Skip on Any Sense Switches Set to 1  
1 0 0 0 0 0 1 0 0 0 0 1 1 1 1 0 (S, R, V mode form)

Skips the next sequential 16-bit halfword if the values of sense switches 1, 2, 3, and 4 are all 1. Leaves the values of CBIT, LINK, and the condition codes unchanged.

Note

This is a restricted instruction.

## INSTRUCTION SETS GUIDE

► VIRY  
Verify                    0311 opcode  
0 0 0 0 0 0 0 0 1 1 0 0 1 0 0 1    (S, R, V mode form)

Executes the verification routine. If there is a failure of any kind, the processor goes on to the next instruction with the number of the test that failed in register A. If there are no errors, the processor skips the next sequential instruction.

If the processor does not have the verification routine, this instruction executes as no-op.

► WCS n  
Writable Control Store  
0 0 0 0 0 0 1 1 1 0 N\6    (R, V, I mode form)

Reserved set of 64 op codes to serve as microcode entrances, where n is 0 through 63.

► XVRV  
XIS Board Verify        1113 opcode  
0 0 0 0 0 0 1 0 0 1 0 0 1 0 1 1    (S, R, V mode form)

XVRV executes a Prime 500 microcode diagnostic routine tht checks the integrity of the XIS board. If the XIS board is not functional, the processor does not skip the next instruction and the A register holds the failed micro-diagnostic test number. If the processor passes the verify instruction, it skips the next instruction.

The codes and tests are:

'72 Data Move Test - Load and Unload XIS Board  
'73 Normalize Test - Adjust Test  
'74 Binary Multiply  
'75 Binary Divide  
'76 Decimal Arithmetic

# F

## 2455 Instruction Sets

The 2455 processor has now been added to the Prime 50 Series computers. This new processor shares the architecture and operating system that is common to all 50 Series processors and makes the 50 Series a line of completely upward-compatible and downward-compatible systems.

The implementation of the common architecture, however, can be slightly different for each member of the 50 Series, allowing the different processors to address a wide variety of user needs while remaining compatible.

The architectural implementation of the 2455 is identical to that of the 2755 processor. This means that instruction set features that apply to the 2755 apply equally well to the 2455. The only exception to this is the STPM (Store Processor Model) instruction: the processor model number code for the 2455 is 32L (decimal).

# SURVEY

READER RESPONSE FORM

Instruction Sets Guide

DOC9474-2LA

Your feedback will help us continue to improve the quality, accuracy, and organization of our publications.

1. How do you rate this document for overall usefulness?

☐ *excellent*      ☐ *very good*      ☐ *good*      ☐ *fair*      ☐ *poor*

2. What features of this manual did you find most useful?

---

---

---

---

---

---

3. What faults or errors in this manual gave you problems?

---

---

---

---

---

---

4. How does this manual compare to equivalent manuals produced by other computer companies?

☐ *Much better*                      ☐ *Slightly better*                      ☐ *About the same*  
☐ *Much worse*                      ☐ *Slightly worse*                      ☐ *Can't judge*

5. Which other companies' manuals have you read?

---

---

Name: \_\_\_\_\_ Position: \_\_\_\_\_

Company: \_\_\_\_\_

Address: \_\_\_\_\_

\_\_\_\_\_

\_\_\_\_\_ Postal Code: \_\_\_\_\_



NO POSTAGE  
NECESSARY  
IF MAILED  
IN THE  
UNITED STATES

First Class Permit #531 Natick, Massachusetts 01760

**BUSINESS REPLY MAIL**

Postage will be paid by:



**Prime**<sup>TM</sup>

Attention: Technical Publications  
Bldg 10  
Prime Park, Natick, Ma. 01760





DOC9474-2LA